UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Bagadus - et fotballanalytisk verktøy

En implementasjon basert på fysiske spillerdata for automatisk generering av statistikk og videosammendrag

Snorre Olderøy Lærum

Masteroppgave våren 2015

Sammendrag

Flere eksisterende systemer for fotballanalyse krever at trenere og andre brukere manuelt må hente og redigere video av kamper. Dette kan medføre et høyt tidsbruk når bestemte kampsituasjoner skal undersøkes og analyseres. I denne oppgaven presenteres et bidrag til en slik problemstilling, der tid og automatisering står helt sentralt.

Vi designer og implementerer en prototype av et post-kamp analyseverktøy. Prototypen er installert på Alfheim stadion i Tromsø og integrerer Bagadus' eksisterende sensor- og videosystem. Sensorsystemet produserer informasjon om spillernes posisjoner og bevegelser på banen under kampene, og fem kameraer under stadiontaket tar opp video. Prototypen inneholder optimaliserte spørringer mot Bagadus-databasen og returnerer statistikk og tidspunkt i kamper hvor bestemte hendelser oppstår. Tidspunktene brukes deretter til å hente video. Vi viser i denne oppgaven hvordan disse spørringene genererer videosammendrag i løpet av sekunder, et aspekt som muliggjør gjennomføring av hurtig og effektiv kampanalyse.

Innhold

Figurer

Tabeller

Forord

For å gjennomføre et godt masterstudium, er det essensielt at det jobbes med et emne som studenten er interessert og engasjert i. Jeg har i denne oppgaven vært så heldig at jeg har fått kombinere fotball og systemutvikling, to felt som interesserer meg i stor grad. Jeg vil gjerne takke min veileder Pål Halvorsen for denne muligheten og all støtten han har gitt meg det siste året. Jeg vil også takke Asgeir Mortensen for hjelp med tekniske aspekter ved oppgaven min.

I løpet av det siste året har jeg tilbrakt mange timer i niende etasje på Institutt for informatikk. Det har vært lange kvelder og flere frustrerende stunder, men takket være mine flotte medstudenter har jeg holdt motet oppe. Flere runder med kortspillene Idiot og President har blitt spilt, og det har vært en fin tid som jeg kommer til å savne. Jeg vil også takke Dana Bakeri som har servert meg med ferske børeker hver eneste dag.

Jeg vil takke Tromsø ILs trenerteam for et utrolig kjekt samarbeid. Dere har vist en entusiasme som har inspirert og motivert meg til å gjøre en god jobb.

Til slutt vil jeg takke venner og familie for hjelp, støtte og oppmuntringer. Mine to samboere Heine og Haldor har til tider holdt ut med en gretten og sliten bergenser, men de har vært oppmuntrende mot meg hele veien. Og en særlig takk til pappa og min søster Henriette som orket å korrekturlese oppgaven min.

Kapittel 1

Innledning

1.1 Motivasjon og bakgrunn

Formasjoner, defensiv plassering, angrepsmønstre og posisjonering av spillere er elementer som fotballtrenere streber etter å perfeksjonere. En treners oppgave er å skreddersy taktiske tilnærminger som drar fordel av sitt eget lags ferdigheter samtidig som at motstandernes svakheter utnyttes. Dette understøtter det overordnede målet om å skåre flere mål enn motstanderne og dermed vinne kampen.

Kampforberedelser inkluderer taktiske vurderinger og beslutninger. Dette aspektet er i stadig utvikling, og fotballklubber over hele verden bruker store ressurser på analysering av kamper og treninger, både for læring, forberedelser og refleksjon av prestasjoner. Videoavspilling av kamper med manuell plukking av avgjørende og kritiske situasjoner, kan argumenteres for å være et primitivt verktøy for analysering. Kostnadene er små, men håndtering av rå-opptak ved hjelp av funksjonaliteter som spoling og pausing, er ineffektivt.

Det eksisterer en rekke fotballanalytiske systemer som bidrar til å gjøre analyseprosessen mer produktiv. Interplay Sports [69], Longomatch [20] og Tacticalpad [13] behandler videostrømmer via manuell analysering og annotering. Andre systemer som Prozone [65], Matchanalysis [28] og SportVU [39] automatiserer en del av den manuelle annoteringsprosessen og bruker kameraer til å samle data om hvordan spillerne beveger seg på banen. I [76] omtales imidlertid denne metoden for innsamling av spillerdata, som upresis og ressurskrevende.

Flere av de eksisterende fotballanalytiske systemene forutsetter manuell integrering mellom videosystemene og kampstatistikk-systemene. De mangler dermed mulighet for auto-

matisk generering av videosammendrag basert på spilleres posisjoner og bevegelser. Dette kan føre til at trenere og andre brukere må interagere med datamaskiner i opptil flere timer for å hente ut bestemte kampsituasjoner. Systemet Bagadus er konstruert som et alternativ til eksisterende løsninger og integrerer et sensorsystem, fotballanalytiske annotasjoner og videoprosessering. All informasjon lagres på samme plass, og dette muliggjør ekstrahering av informasjon på tvers av de ulike subsystemene. Per 2015 er systemet installert på Alfheim stadion, hjemmebanen til Tromsø IL som spiller i den norske tippeligaen. I tillegg pågår installering på Ullevål stadion, beregnet for bruk av det norske herrelandslaget [41, 79, 76, 27].

Bagadus' sensorsystem produserer informasjon om spillernes posisjoner og bevegelser på banen. Denne informasjonen hentes fra sensorbelter levert av ZXY Sport Tracking [82]. Tromsø ILs spillere har disse beltene rundt midjen under kampene, og informasjonen som produseres lagres 20 ganger per sekund i en relasjonsdatabase. Videre filmes kampene av fem kameraer som er montert under stadiontaket på Alfheim stadion. Systemet lagrer opptakene i klipp som er tre sekunder lange, og en hel kamp på 90 minutter vil derav bestå av omtrent 1800 videofiler.

Ved hjelp av *spørringer* mot Bagadus-databasen kan statistikk om spillernes posisjoneringsog bevegelsesmønstre ekstraheres. Systemet muliggjør også uthenting av tidspunkt i kampene hvor spesifikke situasjoner oppstår. Eksempelet som vises i listing 1.1 finner alle situasjoner hvor spiller X befinner seg innenfor venstre 16-metersboks på banen i en bestemt kamp. Tidspunktene som returneres kan deretter brukes til å hente video av hendelsene.

Listing 1.1: Eksempel på spørring mot Bagadus-databasen

```
SELECT player, time
FROM Match
WHERE pos_x < 16,5 AND
pos_y BETWEEN 20 AND 50 AND
player = X
```

1.2 Problemstilling

Flere av dagens verktøy og hjelpemidler for fotballanalyse krever at brukerne må interagere i opptil flere timer med råklipp eller eksisterende systemer for hente ut statistikk og video av bestemte situasjoner. Noen systemer gjør forsøk på automatisering, men disse er ofte preget av upresise og ressurskrevende teknologier. Systemet Bagadus er et alter-

nativ til disse løsningene og integrerer et sensorsystem, fotballanalytiske annotasjoner og videoprosessering. Sensorene som spillerne har rundt midjen under kampene, produserer verdier for følgende attributter:

- tid og dato i sentraleuropeisk tid
- spilleridentifikator
- sensoridentifikator
- spillerposisjon i meter, oppgitt i verdier på x- og y-aksen
- spillers løpsretning
- retning spiller står vendt mot
- spillers løpshastighet i meter per sekund
- energi brukt siden sist måling
- antall meter spiller har løpt så langt i kampen

Disse dataene lagres 20 ganger per sekund i en relasjonsdatabase. Ved hjelp av spørringer mot denne databasen, kan tidspunkt i kamper hvor spesifikke hendelser oppstår, ekstraheres automatisk. De returnerte tidspunktene kan deretter brukes til å hente video av situasjonene. Mortensen skriver i [41] at Bagadus mangler brukergrensesnitt. Dette medfører at brukerne må skrive spørringer selv for hente ut bestemt informasjon. Fotballtrenere flest innehar imidlertid ikke tilstrekkelig kunnskap om SQL[1] til å skrive databasekode. I stedet er det naturlig at fagkyndige konstruerer disse spørringene med parametre som trenerne selv bestemmer via et brukergrensesnitt.

I denne oppgaven skal vi undersøke hvilke muligheter Bagadus presenterer for analyse av kamper og spillere via databasespørringer. Et analyseverktøy beregnet for bruk i etterkant av kampene skal implementeres, og dette skal inneholde ferdigkonstruerte spørringer som henter statistikk og videosammendrag fra kamper. Disse spørringene skal optimaliseres med mål om tilby brukerne mulighet til å utføre hurtig og effektiv kampanalyse.

1.3 Omfang og begrensninger

Denne oppgaven vil ikke gi en fullstendig bakgrunnsbeskrivelse av Bagadus, da dette har blitt gjort tidligere i [27, 79]. Oppgaven vil heller ikke vise personinformasjon om Tromsø ILs spillere. I stedet vil skjermskuddene av det implementerte analyseverktøyet vise spilleridentifikasjoner.

[1] Databasespråk (Structured Query Language)

1.4 Forskningsmetode

Basert på design-metodologien som beskrives av ACM Task Force i *Computing as a discipline* [14], skal vi i denne oppgaven designe, implementere og evaluere en prototype av et fotballanalytisk verktøy. Dette verktøyet skal kunne brukes av Tromsø ILs trenere for å analysere kamper og spillere og vil inneholde ferdigkonstruerte spørringer mot Bagadus-databasen som henter statistikk og video av spesifikke kampsituasjoner. Spørringene skal optimaliseres for å returnere resultater på kortest mulig tid og dermed bidra til at brukerne får gjennomført hurtig og effektiv analyse.

1.5 Bidrag og resultater

I denne oppgaven viser vi hvordan Bagadus' integrerte sensor- og videosystem tilrettelegger for automatisk generering av statistikk og videosammendrag basert på fysiske spillerdata. Vi presenterer hvordan dette systemet muliggjør gjennomføring av effektiv kampanalyse, et fraværende aspekt ved flere eksisterende fotballanalytiske systemer.

En prototype av et post-kamp analyseverktøy har blitt implementert, og dette inneholder ferdigkonstruerte spørringer mot Bagadus-databasen. Spørringene returnerer informasjon som er basert på hvordan spillerne posisjonerer og beveger seg under kampene, og de har blitt optimalisert til å returnere informasjon på kortest mulig tid. To eksempler er *Spiller i område* og *Sprinter*. Førstnevnte finner alle situasjoner hvor en bestemt spiller befinner seg innenfor et avgrenset område på banen. Sistnevnte finner alle situasjoner hvor en bestemt spiller gjennomfører et løp basert på kriterier som løpshastighet, løpsretning og løpsvarighet. Disse spørringene returnerer resultater på under et halvt sekund. Avhengig av kompleksiten til de øvrige implementerte spørringene, bruker majoriteten av dem i verste fall sekunder på å generere statistikker og videosammendrag.

Analyseverktøyet er installert på Alfheim Stadion i Tromsø. Tromsø ILs trenerteam har testet systemet og uttrykt stort engasjement for bruk under den kommende tippeligasesongen. I tillegg har vi inngått et samarbeid med det norske herrelandslaget i fotball, og en demo av verktøyet ble presentert for landslagssjef Per Mathias Høgmo under et seminar [80] på Ullevål stadion høsten 2014.

1.6 Oppgavens struktur

I denne oppgaven presenteres et fotballanalytisk verktøy. Ettersom dette verktøyet integrerer Bagadus' video -og sensorsystem, er det hensiktsmessig at leser har kunnskap om selve Bagadus-systemet. På bakgrunn av dette vil Bagadus og dets subsystemer presenteres i neste kapittel, *Bakgrunn*. Tredje kapittel, *Design av et post-kamp analytisk verktøy* beskriver analyseverktøyet som er blitt implementert med valg som er gjort tilknyttet design og teknologier. Fjerde kapittel, *Ekstrahering av kampsituasjoner*, vil beskrive implementerte databasespørringer som finner video av kampsituasjoner, mens femte kapittel, *Ekstrahering og visualisering kampstatistikk* presenterer hvordan statistikk har blitt ekstrahert og visualisert. Resultatene som har blitt oppnådd, presenteres i sjette kapittel *Resultater*, etterfulgt av *Konklusjon*.

Kapittel 2

Bakgrunn

I dette kapittelet introduseres Bagadus, en prototype av et fotballanalytisk system som sikter mot å integrere eksisterende løsninger og muliggjøre *real-time* fremstilling av kamprelaterte hendelser. Kapittelet vil først presentere andre relaterte fotballanalyse-systemer. Deretter presenteres Bagadus og dets sensorsystem, videosystem og notasjonsanalytiske system. Informasjonen som Bagadus' subsystemer produserer, lagres i én database, og dette bidrar til at alle komponentene er integrerte. Denne databasen vil beskrives, etterfulgt av omtale av mangler og begrensninger ved systemet [76, 41, 27].

2.1 Relaterte fotballanalyse-systemer

Manuell videoanalyse

Interplay Sports [69] er et norskprodusert system for kampanalyse, taktikkstyring og spillerspeiding. I dette systemet blir strømmer av videoer behandlet via manuell analysering og annotering ved hjelp av klassifiseringsskjemaer basert på fotballfaglige begreper (ontologi). Det benyttes også i samband med andre idretter som basketball, bandy og amerikansk fotball [76].

Tacticalpad [13] ligner Interplay Sports og innehar funksjonaliteter knyttet til annotering og redigering av video med tegning av diagrammer og andre animerte elementer. Dette systemet kan blant annet brukes til å analysere kampprestasjoner og planlegge treninger.

Longomatch [20] er et multi-plattform videoanalytisk system som muliggjør importering av video av kamper og treninger. Video kan deretter, i likhet med Interplay Sports,

behandles via manuell annotering med eksisterende *tags* eller brukerdefinerte begreper. Utsnitt av video som har blitt annotert kan deretter lagres i en spilleliste og klargjøres for visning.

Opta [44] samler og analyserer live data fra kamper over hele verden, og de leverer til aviser, tv-stasjoner, bookmakere og profesjonelle fotballklubber. Ansatte følger kampene og lagrer informasjon fortløpende i en database. Denne informasjonen konverteres deretter til å meningsfulle statistikker. Opta leverer blant annet til TV2 Norge.

Automatiserte løsninger

SportVU [39] bruker kameraer til å samle data om hvor ballen, spillerne og dommerne befinner seg på banen. Komplekse algoritmer analyserer datastrømmene og konverterer det til pålitelig, nøyaktig og meningsfull informasjon i form av karakteristikker som ballbesittelse, gjennomsnittlig fart, ballberøringer, skuddhastighet og offside-situasjoner. SportVU benyttes i over 2000 fotballkamper over hele verden, inklusiv alle mesterligakampene (UEFA Champions League) de siste fem sesongene [76].

StatSports [75] sin løsning Viper Pod, inkluderer utstyr for monitorering av individers posisjoner og bevegelser og benyttes av blant andre FC Barcelona, Arsenal og Liverpool FC. Hver enhet inneholder akselerometer, gyroskop, kompass, radiosender, GPS-modul og en hjertefrekvens-mottaker. Videre prosesserer og behandler hele systemet GPS-data 10 ganger i sekundet, og informasjonen som produseres kan strømmes live eller lastes ned.

ProZone [65] innehar funksjonalitet for post-kamp analyse og direkteoverført avspilling av kamper og bruker videoanalytiske programmer til å automatisere en del av den manuelle annoteringsprosessen. Blant annet er det i stand til å måle hyppighet av spilleres bevegelsesmønstre i form av løpshastighet og posisjonering. Prozone brukes av over 300 klubber og organisasjoner på verdensbasis, deriblant den engelske fotballklubben Manchester United.

GPSports [25] leverer små og kompakte GPSer som muliggjør *tracking* av individers posisjoner og bevegelser. Enhetene inkluderer akselerometer og et bånd for toveis trådløs kommunikasjon, og dataene som produseres lagres automatisk i en database.

VX Sport [9] ligner GPSports og leverer en løsning som kombinerer bruk av kompass, GPS, sensorer og Google Earth for å gi informasjon om individers posisjoner og bevegelser. Systemet produserer informasjon om blant annet løpshastighet, akselerasjon og

antall meter løpt og tilbyr grafisk visualisering av ulike typer statistikker.

Match Analysis [28] benyttes av et flertall fotballklubber og landslag på verdensbasis og innehar en rekke produkter tilknyttet analyse av video og spillere. Kampene filmes av tre kameraer som er plassert under stadiontaket på banene, og kombinert produserer de et panoramabilde av hele banen. Via dette panoramabildet kalkuleres spillernes posisjoner, og systemet inkluderer funksjonalitet for ekstrahering av kamphendelser basert på spillernes posisjoner, bevegelser og interaksjon med ballen.

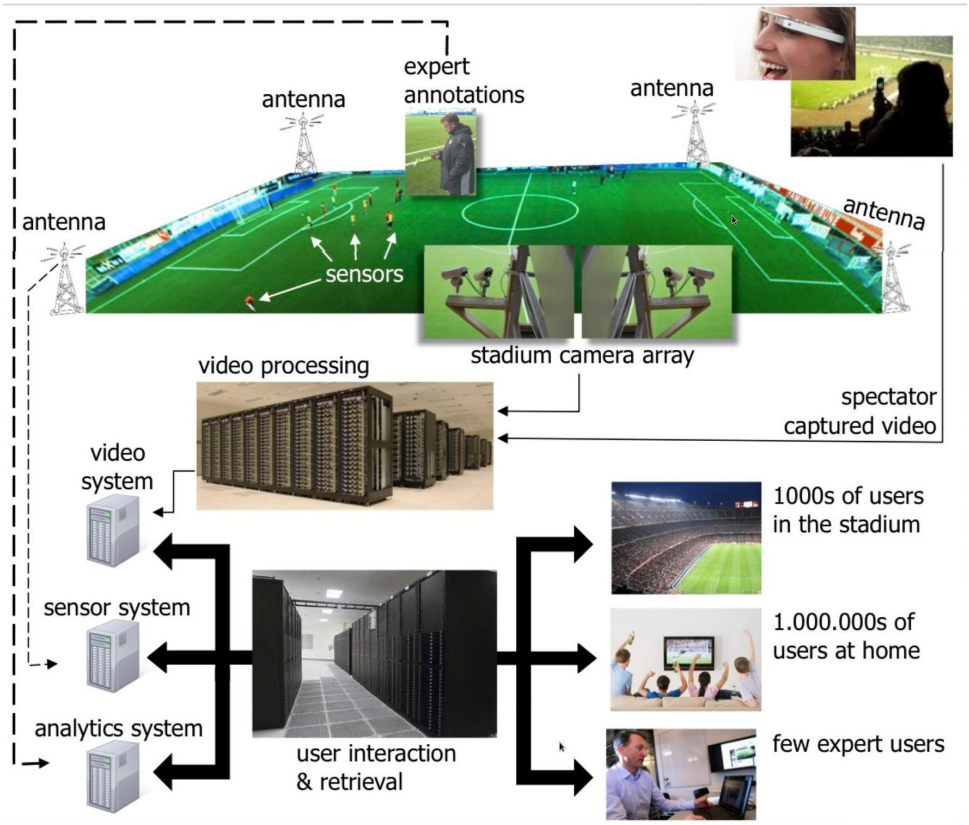
Mangler og begrensninger

Stensland refererer i [76] til flere av de nevnte fotballanalytiske systemene og skriver "...there exist several tools for soccer analysis. However, to the best of our knowledge, there does not exist a system that fully integrates all these features.". Systemer som Interplay Sports, Tacticalpad og Longomatch er basert på manuell uthenting og annotering av bestemte kampsituasjoner. I disse systemene eksisterer det ingen automatisk integrasjon mellom video og fysiske spillerdata, og ekstrahering av hendelser forutsetter at bruker ser gjennom hele videofiler. I kontrast til disse, finnes GPSports og VX Sport. De leverer sensorbaserte løsninger men mangler funksjonalitet for automatisk kobling mellom sensorinformasjon og eventuelle videoopptak. Flere systemer gjør imidlertid forsøk på integrering mellom video og fysisk data. SportVU, Prozone og Match Analysis bruker kameraer til å samle data om spillernes posisjoner og bevegelser på banen, og disse dataene konverteres deretter til meningsfulle statistikker. I [76] beskrives imidlertid denne metoden for innsamling av spillerdata, som upresis og ressurskrevende.

De fleste fotballanalyse-systemer forutsetter manuell integrering mellom videosystemene og kampstatistikk-systemene og mangler mulighet for automatisk generering av videosammendrag basert på spilleres posisjoner. Noen av disse gjør forsøk på automatisering, men uthenting av bestemte kamphendelser er ofte en prosess som krever tid og ressurser. Bagadus er et alternativ til eksisterende løsninger og integrerer et sensor-, video- og notasjonsanalytisk system. Informasjonen som disse tre subsystemene produserer lagres i samme database, og dette muliggjør uthenting av video basert på fysiske spillerdata og annoterte hendelser. Dette systemet presenteres i sin helhet i neste seksjon.

2.2 Bagadus

Bagadus er en prototype av et fotballanalyse-system, konstruert i et samarbeid mellom universitetet i Tromsø, universitetet i Oslo, ZXY Sport Tracking [82], Simula Research Laboratory og Tromsø Idrettslag. Det integrerer et sensorsystem, fotballanalytiske annotasjoner og videoprosessering og søker muliggjøring av direkteoverført fremvisning av kamprelaterte hendelser. Per 2015 benyttes dette på Alfheim stadion, hjemmebanen til Tromsø IL som spiller i den norske Tippeligaen. I tillegg pågår installering på Ullevål stadion, beregnet for bruk av det norske herrelandslaget. Den overordnede arkitekturen vises i figur 2.1 og presenterer tre subsystemer i form av et videosystem, et sensorsystem og et notasjonsanalytisk system. Disse integrerte komponentene resulterer i ett system hvor alle operasjoner håndteres automatisk og alle dataene er tidssynkroniserte [41, 79, 76, 27].



Figur 2.1: Systemoppsett på Alfheim [41]

2.2.1 Videosystemet

Den nåværende versjonen av videosystemet inkluderer en matrise av 5 industrielle kameraer, levert av Basler [7]. Matrisen "dekker hele fotballbanen med tilstrekkelig overlapp

for å identifisere felles funksjoner som er nødvendige for kamerakalibrering og sammenslåing av bilder" [76]. Hvert kamera leverer 50 bilder per sekund (frames per second) over Gigabit Ethernet og støtter en oppløsning på 2046*1086 piksler. Videre er kameraene montert i et 90 graders sirkulært mønster under stadiontaket, og hvert av dem dekker 66 grader av banen [24].

Figur 2.2: Kameramatrisen [23]

Kameraene er synkronisert via et eksternt triggersignal. Dette muliggjør sammenslåing av bilder som sammen produserer panoramabilde av banen. Håndtering av aspekter som lagring og bildesynkronisering gjøres via Bagadus eget bibliotek, Northlight, som integrerer Baslers SDK (software development kit), videokoding via x264 [84] og fargeområdekonvertering via FFmpeg [19, 76].

Videosystemet kan hente, prosessere og levere video av situasjoner basert på tidsintervaller eller spillerposisjoner. Videre finnes det to subversjoner av systemet; et *real-time*-system som støtter direkteoverført avspilling av kamper og et *non-real-time*-system som støtter avspilling av video som er lagret på disk. Den tidligere versjonen av videosystemet, som presenteres i [76], støttet to avspillingsfunksjoner:

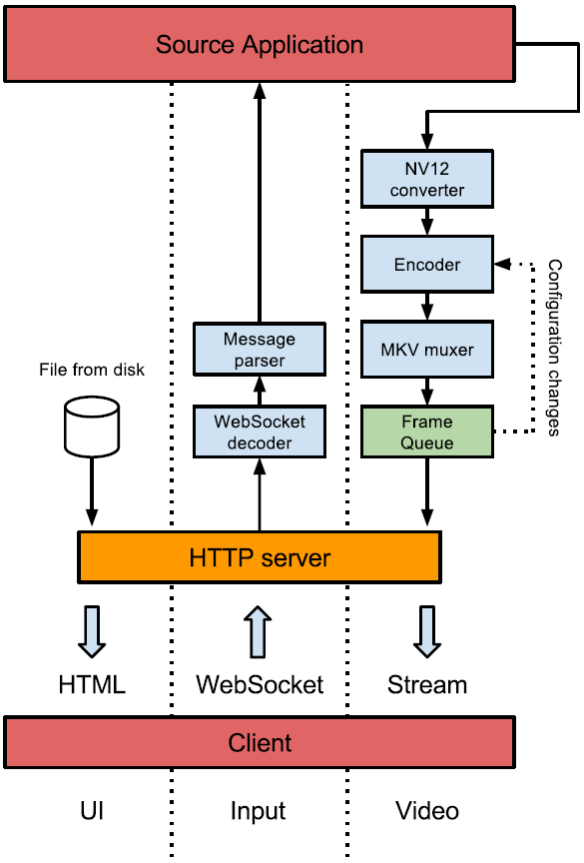
- Avspilling av video ved å manuelt velge et kamera eller ved automatisk bytting av kamera basert på den eller de spillerne som følges.
- Avspilling av video som viser panoramabilde av hele banen

Virtuelt kamera

Den nåværende versjonen av systemet bruker det sylindriske panoramabildet til å generere et virtuelt kamera. Videre finnes det to avspillings-moduser; en manuell og en automatisk. Det manuelle alternativet gjør det mulig for brukerne å styre selv hvilke områder av banen som skal vises i videoavspilleren. Ved hjelp av zooming og panorering av bildene som produseres av de fem kameraene, kan det virtuelle kameraet flyttes overalt på banen, og området som vises kan dekke hele banen eller en størrelsesdefinert sone (se figur 2.4). Den automatiske modusen muliggjør *tracking* av spillere i bildet, og flytting av kamera

er basert på hvor spillerne befinner seg på banen [76, 23, 24, 22].

Videofilene som inneholder en hel kamp i panorama, krever 6GB med komprimert H264 video. For å kunne strømme kampene over internett, må derfor filene komprimeres. I tillegg kreves det også at en CPU[1] eller GPU[2] dekoder 4450*2000 videostrømmer i panorama. Løsningen som presenteres i [93] går ut på å håndtere all logikk på serversiden, og dette gjør at klientsiden kun trenger støtte for WebSocket[3] og HTML5 Video for å kunne hente og spille video.

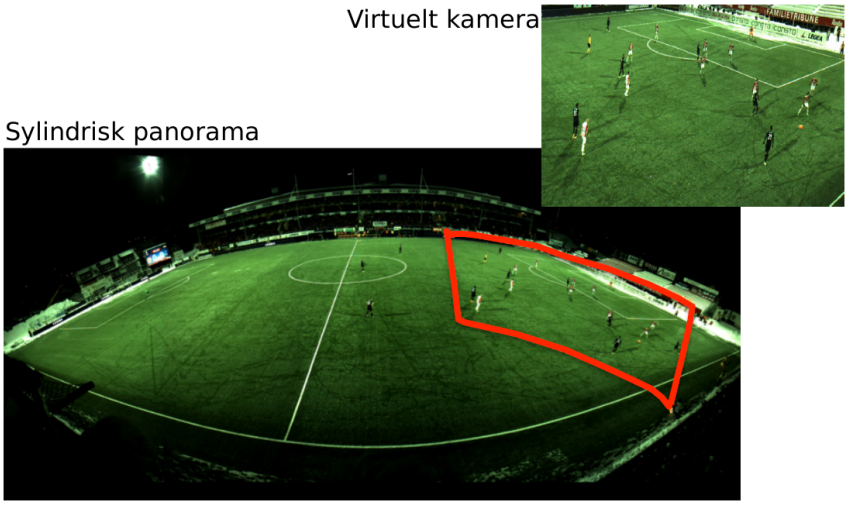


Figur 2.3: Komponenter for generering av virtuelt kamera [93]

Figur 2.3 viser de ulike komponentene som inngår i prosessen om å produsere et virtuelt kamera. Systemet bruker NVENC [95] til å kode H.264-strømmene, og fargebufferet prosesseres av en *OpenGL Compute Shader* [43] som konverterer utdataen til NV12 som igjen fungerer som inndata når NVENC genererer H.264-strømmene. Til slutt signalpak-

[1]Central processing unit: Enhet som håndterer logikk og kalkulering
[2]Graphics Processing unit: Enhet som håndterer grafiske kalkuleringer
[3]En protokoll for gjensidig kommunikasjon mellom kanaler over én TCP-tilkobling(Transmission Control Protocol) [35]

kes utdataen i et MKV multimediaformat. Klienten kan deretter kobles til en standard HTTP-server som returnerer en HTML-side med Javascript-kode som etablerer en WebSocket mellom klienten og serveren. Dette resulterer i at HTML5 videoelementet kan spille av H.264-videostrømmen [93, 24].



Figur 2.4: Virtuelt kamera [23]

2.2.2 Sensorsystemet

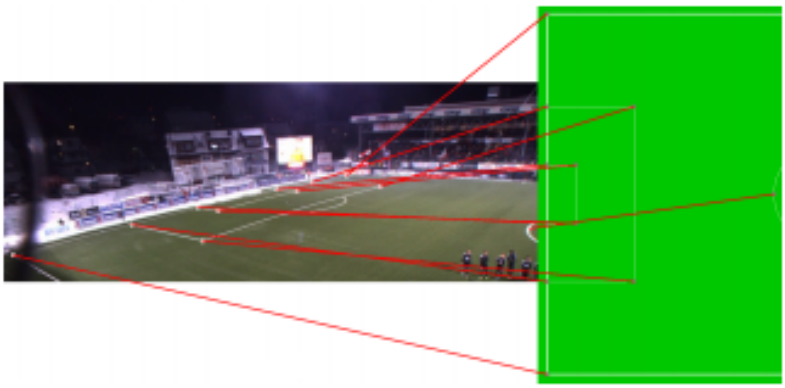
Sensorsystemet gir informasjon om spillernes posisjoner og bevegelser på banen. Denne informasjonen hentes fra sensorbelter levert av ZXY Sport Tracking [82], som Tromsø ILs spillere har rundt midjen under kampene. ZXY leverer også 11 stasjonære radiomottakere som er plassert på tribunetaket rundt Alfheim stadion. Hver radiomottaker har et mottaksområde på 90 grader og måler spillernes posisjoner på banen ved vektorbasert prosessering. Den nåværende versjonen av sensorsystemet er basert på et 20 GHz ISM bånd for radiokommunikasjon og signaloverføring, og sensorene bruker et akselerometer som registrerer spillernes bevegelser i alle tre direksjonale akser. I tillegg består de av pulsmålere og gyroskop som i kombinasjon med kompass, muliggjør *tracking* av spillernes løpsretninger. Sensordataene overføres i *real-time* til en relasjonsdatabase, og systemet muliggjør tidssynkronisering av all data under lagring i databasen. Dette gjøres ved å samle all sensorinformasjon i samme radiosignal som brukes for å måle spillerposisjoner [54, 27].

Informasjonen som sensorene produserer, lagres 20 ganger i sekundet i databasen, og en kamp på 90 minutter produserer ca 2.4 millioner dataelementer. Sensordataene inkluderer verdier for følgende attributter:

- tid og dato i sentraleuropeisk tid
- spilleridentifikator
- sensoridentifikator
- spillerposisjon i meter på banens x- og y-akse
- retning spiller løper mot. Verdien oppgis i radianer[4] hvor 0 er y-aksens retning. Hvis spiller løper mot venstre på banen (fra kameraenes perspektiv), er verdien mindre enn 0 og større en -3,14. Hvis spiller løper mot høyre er verdien større enn 0 og mindre enn 3,14.
- retning spiller står vendt mot. Verdien oppgis i radianer hvor 0 er y-aksens retning.
- spillers løpshastighet i meter per sekund
- energi spiller har brukt siden sist måling
- antall meter spiller har løpt så langt i kampen

Sensorbeltene er små og kompakte og medfører ingen ekstra belastning for spillerne under kampene. I tillegg har FIFA godkjent bruk av beltene under internasjonale kamper [76, 54].

Spillernes posisjoner på banen representeres ved hjelp av det kartesiske koordinatsystemet [55], og kameraene er montert inne i et kontrollrom som er plassert under taket midt på stadion. Lokalisering av spillere i video avhenger av at sensorkoordinatene konverteres til bildepiksler. Halvorsen skriver i [27] at de i den forbindelse kalkulerte og tok i bruk en 3*3 transformasjonsmatrise som *mapper* punkt i bildeplanet ved hjelp av kjente punkt på banen (se figur 2.5) [54].



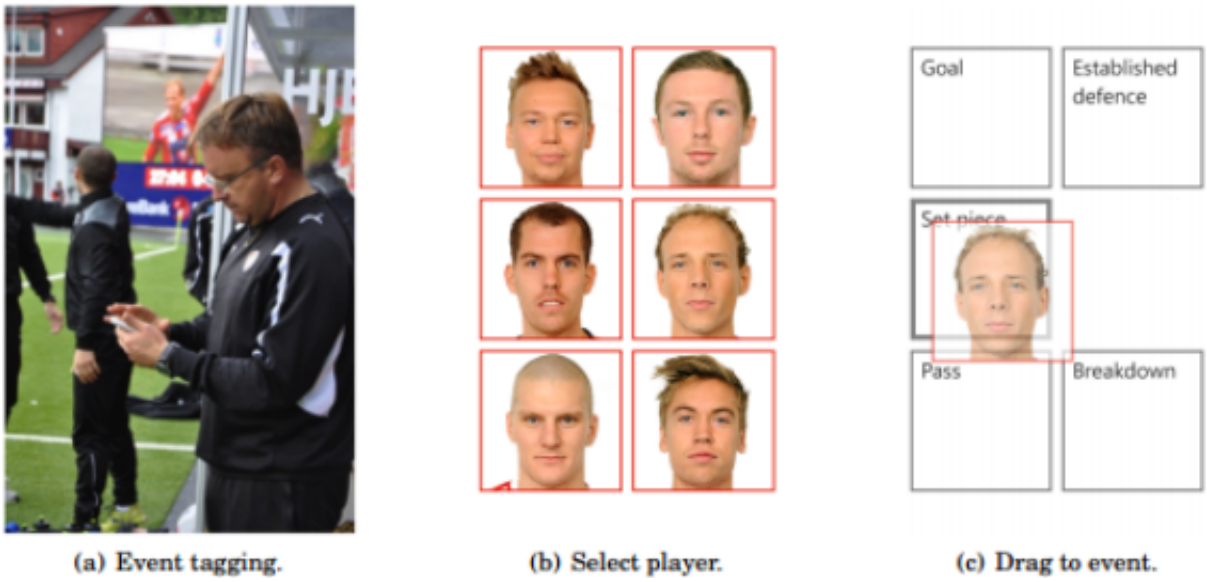
Figur 2.5: Mapping mellom ZXY og bilde [27]

[4] 1 radian = 57,32 grader

2.2.3 Det notasjonsanalytiske systemet

Muithu [33] er et notasjonsanalytisk system med formål om å begrense alt manuelt arbeid knyttet til identifisering av bestemte kamphendelser. Det kan aksesseres via en smarttelefon eller et nettbrett og gjør det mulig for trenere å registrere forhåndsdefinerte kamphendelser. De registrerte hendelsene lagres deretter i en database og kan når som helst hentes ut og vises i et videoformat. Brukergrensesnittet til Muithu er basert på drag-and-drop-funksjonalitet og er skissert i figur 2.6 [76].

I forbindelse med en annen pågående masteroppgave, implementeres det en ny og mer moderne notasjonsanalytisk løsning. Dette systemet innehar funksjonaliteter for *tagging* av kamphendelser og avspilling av og tegning på video i *real-time* via Bagadus' virtuelle kamera.



(a) Event tagging. (b) Select player. (c) Drag to event.

Figur 2.6: Definering av kampsituasjoner via Muithu [76]

2.2.4 Databasen

Bagadus-databasen er en PostgreSQL-database [26] og er lokalisert på en server på Alfheim stadion i Tromsø. Den inneholder tabeller med informasjon fra videosystemet, sensorsystemet og det notasjonsanalytiske systemet og er designet på en hensiktsmessig måte for integrering mellom de tre subsystemene. Databasen omfatter skjemaer for blant annet kampene, banen(e), lagene, enkeltspillerne, *events* tagget via Muithu [33] og video som produseres av videosystemet. Ettersom all informasjonen er lagret på samme plass, foreligger det store muligheter knyttet til henting av informasjon på tvers av subsystemene.

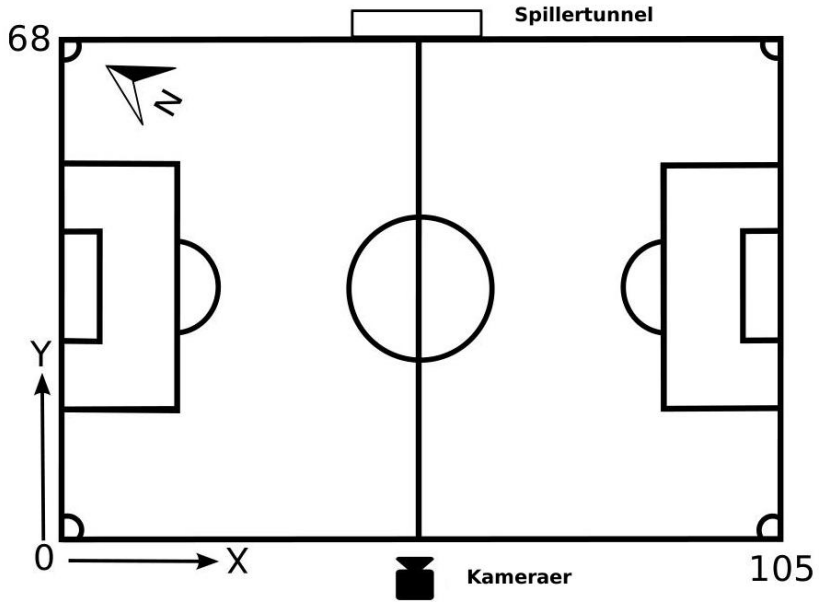
Under kampene blir dataene som genereres av ZXY-sensorene, lagret 20 ganger per sekund i en egen kamptabell i databasen (se figur 2.8). Denne informasjonen inkluderer og har følgende format [54]:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid.
- **field_id** (int) - Identifikator for banen kampen spilles på.
- **tag_id** (int) - Sensoridentifikator.
- **player_id** (int) - Spilleridentifikator.
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen i meter.
- **facing** (float) - Retning spiller står vendt mot. Verdien oppgis i radianer, og 0 er y-aksens retning.
- **direction** (float) - Retning spiller løper mot. Verdien oppgis i radianer, og 0 er y-aksens retning.
- **energy** (float) - Estimert bruk av energi etter sist måling. Måleenheten er udefinert og relativ til hvert individ
- **velocity** (float) - Spillers løpshastighet i meter per sekund.
- **total_distance** (float) - Antall meter spiller har løpt så langt i kampen.

Spillerposisjonene oppgis i todimensjonale koordinater som er kalibrert til banen på Alfheim stadion. Posisjonen (0,0) er lokalisert i det nordvestlige hjørnet som er nederst til venstre fra kameraenes perspektiv. Oppsettet vises i figur 2.7.

Parallelt med sensorsystemet, lagrer videosystemet video i form av en samling av tresekunders segmenter. En kamp på 90 minutter vil derfor bestå av over 1800 videofiler. Informasjonen om disse filene lagres samlet i en tabell, *field_recorded*, og har følgende format [41]:

- **field_id** (int) - Identifikator for banen som kampen spilles på
- **file_id** (int) - Filidentifikator
- **file_size** (float) - Størrelse på filen
- **file_start** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **file_uri** (String) - Sti og filnavn



Figur 2.7: Koordinater på Alfheim Stadion [54]

field_id	player_id	timestamp	position
1		2013-11-03 18:01:09	(26.5727,29.4356)

tag_id	facing	direction	energy	velocity	total_distance
2	0.7998	0.8241	150.6617	0.9676	255.58430002718

Figur 2.8: Utsnitt av sensordataene fra Tippeligakampen mellom Tromsø og Strømsgodset i 2013

2.2.5 Automatisk genering av statistikk og video basert på spilleres posisjoner og bevegelser

Hver kamp som lagres, representeres av en egen kamptabell i databasen. Disse tabellene inneholder informasjon om spillernes posisjoner og bevegelser på banen i den gitte kampen. I tillegg inneholder samme database informasjon om video som har blitt tatt opp. Ettersom all data lagres samme plass, foreligger det store muligheter for ekstrahering av informasjon på tvers av Bagadus' subsystemer. Innholdet i og designet av databasen legger derfor til rette for uthenting av statistikk og video basert på spesifikke spilleres posisjonerings- og bevegelsesmønstre.

Ved hjelp av *spørringer* mot databasen, kan bruker ekstrahere informasjon basert på et sett med betingelser. Listing 2.1 og 2.2 er to eksempler på slike spørringer og finner henholdsvis informasjon om hvor langt hver spiller har løpt i en kamp og tidspunkt hvor en

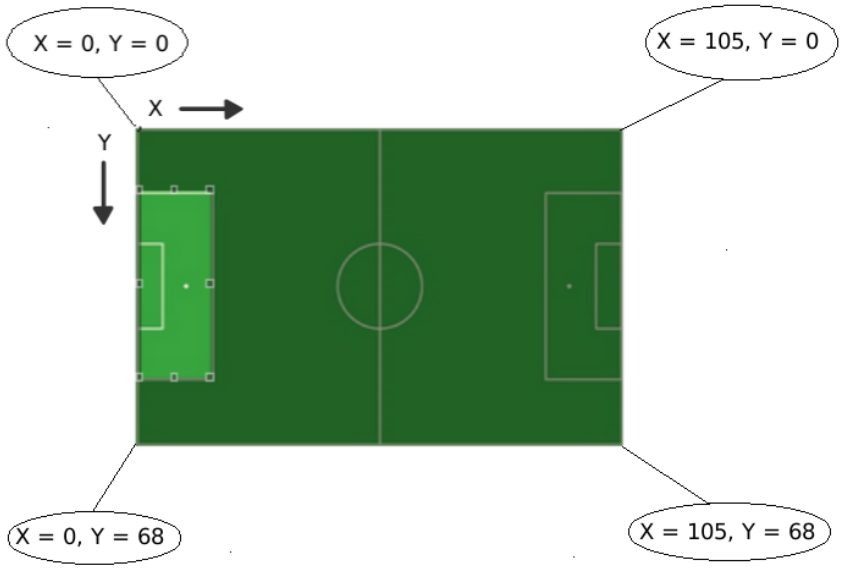
bestemt spiller befinner seg i en av 16-metersboksene.

Listing 2.1: Spørring som finner den totale avstanden hver spiller har løpt

```
SELECT player_name, max(total_distance)
FROM Match x
GROUP BY player_name;
```

Hvor langt spillerne har løpt underveis i kampene lagres fortløpende i form av attributtet *total_ distance*, og avstanden vil øke så lenge spillerne er på banen. Derfor benyttes SQL sin innebygde funksjon *max()* [89] (max(total_distance)) for å finne den totale avstanden.

Spørringen som presenteres i listing 2.2 finner alle situasjoner hvor spiller X befinner seg innenfor 16-metersboksen på venstre siden av banen (fra kameraenes perspektiv). Betingelsene tilknyttet denne lokasjonen defineres i en *where*-klausul [90] og spesifiserer at den gitte spilleren må befinne seg mindre enn 16.5 meter fra den venstre dødlinjen og mellom koordinatene 17.5 og 50.5 på y-aksen. Dette området visualiseres i figur 2.9



Figur 2.9: 16-metersboks

Listing 2.2: Spørring som finner situasjoner hvor en spiller er i en av 16-metersboksene [41]

```
SELECT (timestamp -10 sec) as start,
(timestamp + 10 sec) as stop
FROM Match x
WHERE x_pos > 0 AND x_pos <16.5
```

```
AND y_pos > 17.5 AND y_pos < 50.5
AND player = X
ORDER BY timestamp
```

Videosystemet lagrer video i ett tresekunders klipp av gangen, og en hel kamp vil bestå av rundt 1800 videofiler. Når situasjoner skal analyseres, er det hensiktsmessig at videosammendrag inneholder predefinerte start- og sluttidspunkt som viser hva som ledet til at situasjonen oppsto og hva den endte i. Spørringen som presenteres i listing 2.2 viser et eksempel på hvordan dette kan gjøres og definerer et tidsintervall på 20 sekunder (10 sekunder før og etter situasjonen oppstår). Dette intervallet brukes i spørringen i listing 2.3 som henter video.

Listing 2.3: Spørring som henter video

```
SELECT *
FROM field_recorded
WHERE file_start between 'start' AND 'stop'
ORDER BY file_start
```

2.2.6 Mangler og begrensninger

Et promoterende element for Bagadus er løsningen som automatiserer deler av det manuelle arbeidet som sportsanalyse krever i de eksisterende systemene. Ved hjelp av databasespørringer, kan informasjon om spesifikke kampsituasjoner ekstraheres, og dette bidrar til å forkorte tiden fotballanalytikere må tilbringe foran en datamaskin.

Mortensen skriver i [41] at "At the time of the writing, we do not have any logical interface, i.e., the users must write traditional SQL queries. Our target users will probably not have such knowledge, and more intuitive, user-friendly interfaces are currently being researched.". Fotballtrenere flest innehar ikke tilstrekkelig kunnskap om SQL til å skrive databasekode selv. For å kunne gjøre systemet anvendelig for brukere uten informatisk bakgrunn, er det derfor behov for et verktøy som konstruerer spørringer for dem, og disse spørringene bør være basert på parametre som enkelt kan defineres i et brukergrensesnitt.

2.3 Sammendrag

I dette kapittelet har Bagadus, en prototype av et fotballanalyse-system, blitt introdusert. Dette systemet er installert på Alfheim stadion i Tromsø og integrerer et sensorsystem,

fotballanalytiske annotasjoner og videoprosessering. De tre subsystemene produserer informasjon som lagres i samme database, og ved hjelp av spørringer kan statistikk og video av kampsituasjoner, ekstraheres.

Per 2014 mangler systemet intuitive brukergrensesnitt, og dette medfører at brukerne må skrive databasekode selv for å ekstrahere statistikk og video av kampsituasjoner. Systemets brukergruppe er derfor begrenset til personer med teknisk bakgrunn. Det er rimelig å anta at trenere og spillere ikke innehar tilstrekkelig kunnskap om SQL til å skrive spørringer selv. På bakgrunn av dette er det behov for et post-kamp analytisk verktøy som konstruerer spørringer basert på argumenter som brukerne kan definere i et brukergrensesnitt. I neste kapittel vil en prototype av et slikt system presenteres.

Kapittel 3

Design av et post-kamp analytisk verktøy

I dette kapittelet introduseres et post-kamp fotballanalytisk verktøy som integrerer Bagdus' video- og sensorsystem. Kapittelet vil redegjøre for motivasjon, krav til systemet, design og teknologier som har blitt benyttet.

3.1 Motivasjon

Bagadus muliggjør, via dets sensor- og videosystem, automatisk generering av statistikk og videosammendrag basert på spilleres posisjoner og bevegelser. Dette forutsetter imidlertid kunnskap om databaser og spørringer, et teknisk aspekt som fotballtrenere flest ikke kan noe om. Uten intuitive brukergrensesnitt, begrenses derfor Bagadus' brukergruppe til personer med teknisk bakgrunn.

Formålet med verktøyet som presenteres i denne oppgaven, er å tilby trenere en hurtig og intuitiv måte å hente statistikk og videosammendrag fra kamper på. Verktøyet vil inneholde ferdigkonstruerte spørringer, basert på brukerdefinerte argumenter som defineres i brukergrensesnittet. For å redusere tiden til hver spørring og dermed øke brukervennligheten, har det blitt lagt vekt på effektivitet og optimalisering av spørringene.

3.2 Kravbehandling

Representanter fra Tromsø IL har formidlet idéer til hva et potensielt post-kamp analyseverktøy bør inneholde. På bakgrunn av disse idéene har følgende sett med krav til systemet blitt definert:

Ikke-funksjonelle krav:

- **Tilgjengelighet**: Systemet skal kunne lett aksesseres av trenere og spillere, uavhengig av digital enhet og operativsystem.
- **Responsivitet**: Systemet skal kunne benyttes via datamaskin, smarttelefon og nettbrett.
- **Tilgang til kamper**: Systemet skal ha tilgang til alle hjemmekamper som har blitt spilt i den pågående tippeligasesongen.
- **Responstid**: Video av kampsituasjoner og statistikk skal returneres på kortest mulig tid.

Funksjonelle krav:

- **Statistikk og videosammendrag**: Systemet skal kunne hente og vise statistikk og video av situasjoner fra kamper.
- **Brukerdefinerte argumenter**: Brukerne skal selv kunne bestemme betingelsene for informasjonen som skal hentes.
- **Spilleliste**: Søkeresultat skal samles og vises i en spilleliste.
- **Video**: Video av ønskede kampsituasjoner skal kunne spilles av i systemet.

3.3 Teknologier og verktøy

På bakgrunn av krav som stilles til systemet, ble det gjort et valg angående hvordan systemet skal brukes og hvilke teknologier som skal forme det. Webbaserte løsninger har den fordelen at det kan benyttes via mac-, linux- og windowsmaskiner, samt apple- og androidtelefoner. Slike systemer kan installeres på en server og dermed lett aksesseres via en nettleser.

Målet med prototypen er å tilby brukerne en hensiktsmessig og intuitiv måte å hente statistikk og video av kampsituasjoner på. Ettersom all informasjon er lagret i én database, må derfor systemet tilrettelegge for kommunikasjon mellom en webserver og en databaseserver. I prototypen går denne kommunikasjonen via en applikasjonsserver med komponenter som er kodet i Java. Bakgrunnen for dette har rot i utviklers erfaring med dette programmeringsspråket, og det finnes flere verktøy og rammeverk basert på åpen kildekode, som muliggjør kobling av Javaklasser med databaser og webservere.

Intellij og Bitbucket

I forbindelse med denne oppgaven har programmeringsverktøyet Intellij IDEA [32] blitt benyttet. Dette verktøyet støtter utvikling gjennom programmeringsspråk som blant annet Java, Javascript, HTML, XML og SQL, og bidrar til å opprettholde ryddige prosjektstrukturer. I tillegg støttes integrering med versjonskontrollsystemer. I dette prosjektet har Bitbucket [15] blitt benyttet. Bitbucket tilbyr opplasting av kode til private oppbevaringsservere.

Maven og Jetty

Maven [5] er et prosjekthåndteringsverktøy for utvikling av ulike typer systemer. POM-filen (Project Object Model) er hovedessensen i dette verktøyet og inneholder XML-kode som bidrar til å opprettholde en hensiktsmessig og ryddig prosjektstruktur, i tillegg til å holde rede på prosjektavhengigheter og informasjon om bygging og kjøring av prosjektet. Maven bygger og kompilerer prosjektet ved èn enkelt terminal-kommando, *mvn clean install* [67].

I utviklingsfasen har Jetty [18] blitt benyttet. Dette verktøyet er basert på åpen kildekode og kan opprette en lokal webserver via Maven-kommandoen *mvn jetty:run*. Dette oppnås ved å definere Jetty som en *plugin* i POM-filen, og etter kjøring kan systemet aksesseres via *localhost:8080*.

Spring

Rammeverket Spring [73] har Java som plattform og kan benyttes i samband med alle typer Java-applikasjoner. I tillegg støttes utvikling av webapplikasjoner over denne plattformen. Rammeverket kan bidra til sammenkobling av, kommunikasjon mellom og konfigurering av ulike deler, moduler og objekter. Ved å definere en avhengighet til Spring i POM-filen, lastes biblioteket når prosjektet bygges.

Analyseverktøyet består av et sett med Javaklasser som håndterer logikk og kommunikasjon med databasen. Klassen *MatchController* er annotert *@Controller* og fungerer som et bindeledd mellom frontend-delen[1] av systemet og tjenesten som kommuniserer med databaseserveren. Kontrolleren består av metoder som sender metodeparametrene videre til funksjoner i en klasse annotert *@Service* [72] (*MatchService.java*). Disse metodene konstruerer SQL-spørringer basert på argumentene. Spørringene sendes videre til en gitt

[1]Delen av systemet som håndterer design og presentasjon av informasjon i brukergrensesnittet

database-URL, og resultatet returneres tilbake til kontrolleren. Herfra returneres dette i JSON-format [34], til et REST-API [74]. Frontend-delen av systemet henter resultatene via dette APIet. Ved hjelp av metodeannotasjonen *RequestMapping* [71] (se listing 3.1), bestemmes hvilken URL som resultatene postes til og kan hentes via [68].

Listing 3.1: Eksempel på bruk av RequestMapping og REST-API

```
@RequestMapping(method = RequestMethod.GET,
value = "/data/mostMetersRun/{table_name}",
produces = "application/json")
@ResponseBody
public JSONObject getMostMetersRun(
@PathVariable String table_name) {
  ...
}
```

Argumentene sendes fra frontenden til backenden[2] i selve URLen. Ved å definere parametrene i kontrollermetodene som *@PathVariable* [70] samt inkludere disse i URL-stien, åpnes kommunikasjonen mellom de to delene.

AngularJS

Javascript-rammeverket AngularJS [4] benyttes for å binde brukergrensesnittene med backend-delen av systemet. Ved å kalle på definerte Javascript-funksjoner (se listing 3.2 og 3.3) som videre kaller på URLer som det ovennevnte REST-APIet definerer, oppstår det en kommunikasjon mellom brukergrensesnittene og databasen. Spiller- og kampinformasjon fra databasen kan visualiseres, og parametre kan sendes fra webgrensesnittene til metodene som konstruerer spørringer.

Listing 3.2: Eksempel på Javascript-funksjon som sender kamptabell som parameter til en bestemt URL i REST-APIet

```
this.MatchController = {};
this.MatchController.getPlayers = function(table_name) {
  var req = {};
  req.method = 'GET';
  req.url = prefix + '/data/{table_name}/players';
  req.url = req.url.replace(/{table_name}/, table_name);
```

[2]Delen av systemet som håndterer logikk og kommunikasjon med databasen

```
  req.params = {};
  return $http(req).then(getData);
};
```

Listing 3.3: AngularJS-kode som henter spillerinformasjon

```
BagadusData.MatchController.getPlayers($scope.match).
then(function(data) {
  $scope.players = data;
});
```

Scope er et viktig aspekt ved AngularJS og kan beskrives som 'limet mellom applikasjonskontrolleren og viewet' [3]. Et slikt objekt kan defineres i form av lister, funksjoner eller variabler (se listing 3.4) og kan deretter aksesseres i HTML-koden.

Listing 3.4: Eksempel på bruk av *scope* i AngularJS

```
$scope.selectHalf = function(half) {
  $scope.half = half;
}
```

Applikasjonen bruker AngularJS-funksjonen *routeProvider* [1] til å konfigurere ruting. Ved hjelp av metodene *when* og *otherwise*, bestemmes hvilke HTML-sider som skal returneres basert på hvilke URLer. Et eksempel på dette vises i listing 3.5.

Listing 3.5: Eksempel på bruk av *routeProvider*

```
bagadus.config(function ($routeProvider,
$httpProvider,$locationProvider) {
  $routeProvider.
  when('/bagadus', {
    controller: 'FrontPageCtrl',
    templateUrl: templatePrefix + "frontpage.html"
  }).
  otherwise({redirectTo: '/bagadus'});
});
```

Bootstrap

Brukergrensesnittene i analyseverktøyet er skrevet i HTML5. Dette språket brukes til å strukturere og presentere elementer og informasjon i nettlesere. Responsivitet er et av de ikke-funksjonelle kravene som stilles til systemet, og på bakgrunn av dette har Bootstrap [53] blitt benyttet. Bootstrap er et frontend-rammeverk med åpen kildekode og inneholder ferdigkonstruert kode for styling av HTML-elementer. Ved å kalle på klasser som defineres i Bootstrap, blir HTML-elementene automatisk stylet. Bruk av dette rammeverket har minsket behovet for manuell styling via CSS[3] og bidrar til brukergrensesnitt som er dynamisk skalert og estetisk stylet.

Jcrop

Noen av spørringene som presenteres senere i oppgaven, inkluderer betingelser som angår bestemte områder på banen. For å tilby brukerne av systemet en hensiktsmessig måte å definere et avgrenset område på banen på, har Jcrop [38] blitt benyttet. Dette rammeverket tilbyr mulighet for markering av felt på bilder (se blant annet figur 4.2). Ved å la bildet være en opptegnet og forminsket versjon av banen på Alfheim, kan koordinatene til det markerte området hentes via JQuery [21] og deretter konverteres til koordinater på den ekte banen.

Kommunikasjon med databasen

Informasjonen som Bagadus' subsystemer produserer, lagres i en PostgreSQL-database [26] som er lokalisert på en server på Alfheim stadion i Tromsø. PostgreSQL er et objekt-relasjonelt databasesystem basert på åpen kildekode, og testdatabasen som har blitt benyttet i utviklingsfasen av dette prosjektet, er av samme sort. Et viktig element i denne oppgaven har vært å finne en hensiktsmessig måte å eksekvere spørringer mot databaseserveren på. I den anledning ble rammeverket Hibernate [66] vurdert. Hibernate er et objekt-relasjonelt kartleggingssystem og inkluderer et API[4] som muliggjør lagring og henting av Java-objekter fra en relasjonsdatabase. Dette rammeverket fjerner derfor behovet for å manuelt konvertere databaserelasjoner til Java-objekter.

Et alternativ til Hibernate er Java-APIet JDBC (The Java Database Connectivity) [52] som også muliggjør kobling mellom Java-klasser og databaser. APIet inkluderer klasser for oppretting av databaseforbindelse, samt eksekvering av egenkomponerte spørringer. Ved eksekvering innpakkes resultatene i et objekt, og dette objektet returneres. Deretter

[3] Cascading Style Sheet: Kodespråk for styling av HTML-elementer
[4] Application Programming Interface

må disse resultatene manuelt prosesseres.

Ved bruk av Hibernate, fjernes behovet for å manuelt konvertere databasetabeller til objekter. Det er imidlertid ikke strukturering og presentasjon av objekter og relasjoner som utgjør hovedfokuset i denne oppgaven. Spørringene som presenteres i kapittel 4 og 5 er basert på ulike typer brukerdefinerte betingelser, og de returnerer ulike typer attributter. Det er derfor behov for et verktøy som muliggjør konstruksjon og eksekvering av alle typer spørringer og funksjoner som er basert på brukerinput. På bakgrunn av dette, har JDCB blitt benyttet.

Klassen *DriverManager* [46] kan beskrives som en tjeneste som håndterer JDBC-drivere og tilbyr gjennom den statiske metoden *getConnection()*, tilkobling til en gitt database-URL. Metoden returnerer et *Connection*-objekt [49] som utgjør konteksten hvor SQL-spørringer sendes og resultater returneres.

Etter en database-tilkobling har blitt etablert, kan SQL-spørringer konstrueres og pakkes inn i et *String*-objekt [47]. Eksekvering av spørringer gjøres ved å opprette et *Statement*-objekt [51] i konteksten av en *Connection* og kalle på objektets metode *executeQuery()* med spørringen som parameter. Metoden returnerer et *ResultSet*-objekt [50] som inneholder rader med resultater fra spørringen (se listing 3.6).

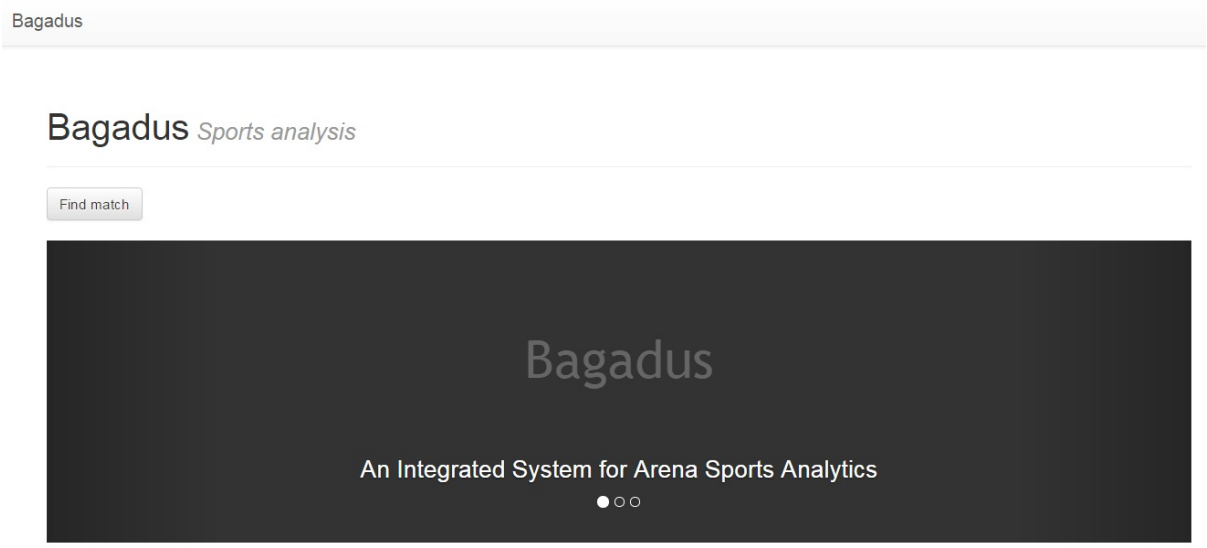
Listing 3.6: Exempel på eksekvering av spørring

```
public ResultSet findTotalDistance(String match) {
  String query =
  ‘SELECT player_name, max(total_distance)\n‘ +
  ‘FROM ‘ + match + ‘\n‘+
  ‘GROUP BY player_name;‘;

  Statement statement = null;
  ResultSet resultSet = null;
  Connection connection = DriverManager.getConnection(
    jdbc, user, password);

  try {
    statement = connection.createStatement();
    resultSet = statement.executeQuery(query);
  } catch (Exception e) {
```

```
            e.printStackTrace();
        }
        return resultSet;
    }
```

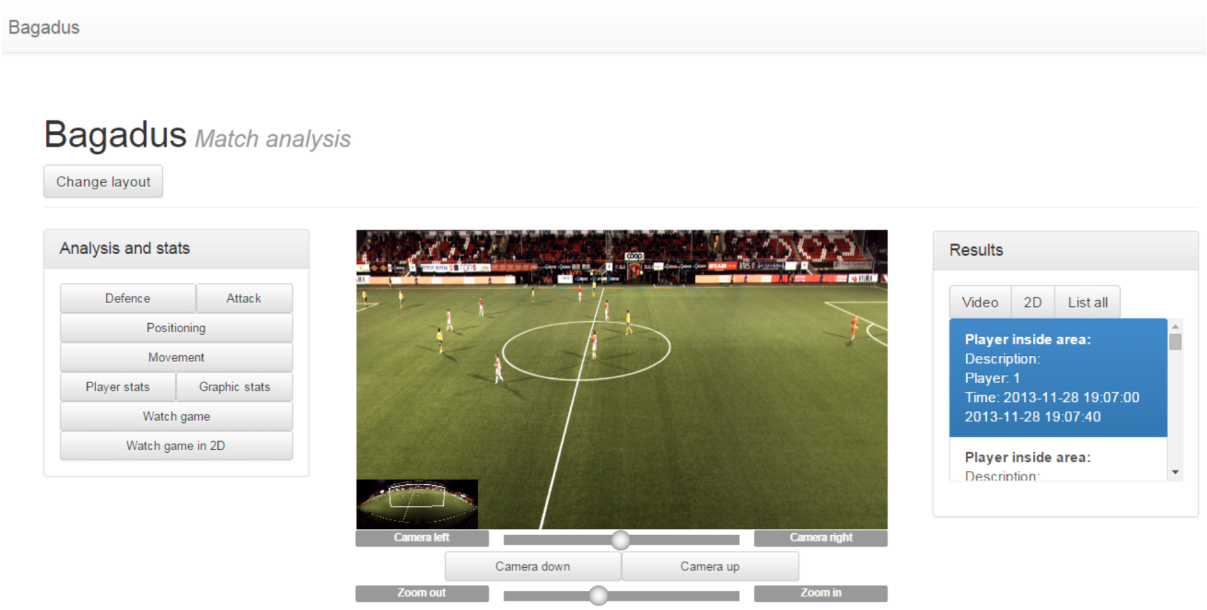
3.4 Design

Applikasjonen initialiseres gjennom en startside (se figur 3.1). Herfra kan bruker navigere seg til et brukergrensesnitt som viser informasjon om kampene som er lagret i databasen. Denne informasjonen inkluderer dato og hvilke to lag som spilte kampen, samt en link som kan viderebringe bruker til kampanalysesiden. Dette brukergrensesnittet består av tre deler; et panel som presenterer ferdigkonstruerte spørringer fordelt i logisk navngitte kategorier, en videospiller og et panel som samler og viser informasjon om kampsituasjonene som har blitt hentet. Denne siden vises i figur 3.2.



Figur 3.1: Startside

Brukergruppen vil utelukkende bestå av spillere, trenere og andre fotballfaglige personer. Det er derfor essensielt at systemet kan tolkes og forstås i henhold til et universalt fotballspråk. Per 2014, og før det har eksistert interagerbare applikasjoner for Bagadus, har Interplay Sports [69] blitt benyttet av analyseansvarlige ansatte i Tromsø Idrettslag. Dette systemet behandler strømmer av videoer via manuell analysering og annotering ved klassifiseringsskjemaer som er basert på allment kjente fotballbegreper. Panelet (se figur 3.2) som viser hvilke spørringer som er implementert og kan brukes, er inspirert av Interplay

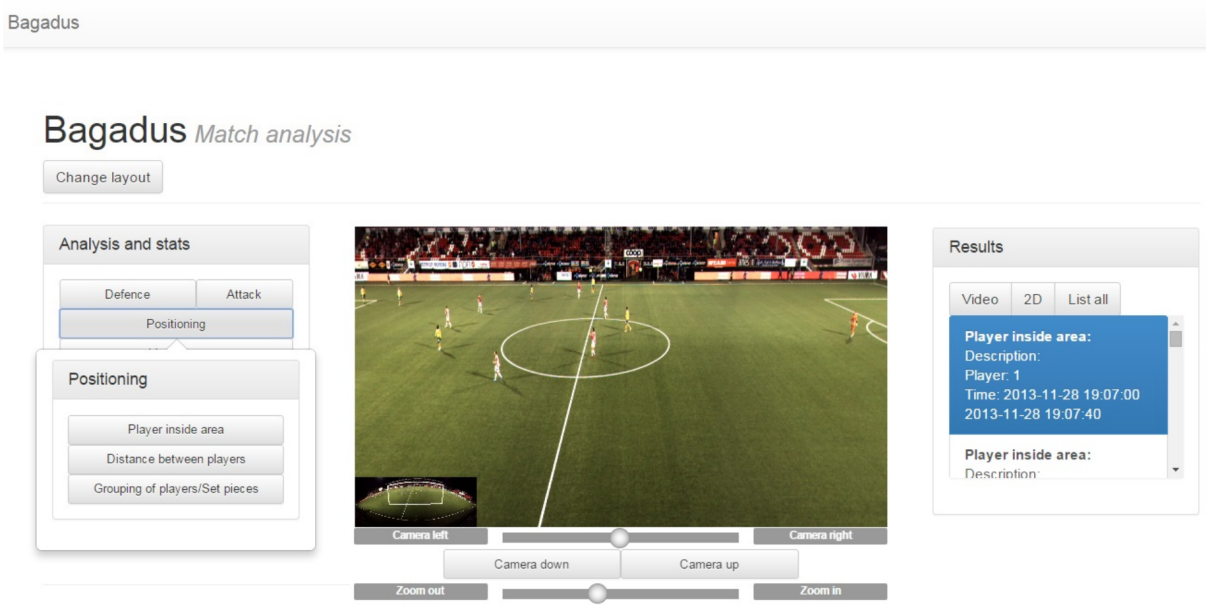


Figur 3.2: Brukergrensesnitt for kampanalyse

Sports' tilnærming til bruk av slike fotballbegreper og består av diverse kategorier i form av knapper. Ved å trykke på en av disse, vises et tooltip[5] bestående av et nytt sett med knapper som representerer selve spørringene. Kategoriene er navngitt med utgangspunkt i generelle og overordnete begreper knyttet til fotball og analyse, mens spørringene har titler som korresponderer til de konkrete kampsituasjonene som spørringene finner. idéen er at spørringer som for eksempel er tilknyttet det defensive spillet skal samles et sted, mens de som omhandler generell posisjonering av spillere havner i en annen kategori (se figur 3.3) [76].

Hver spørring representeres av knapper i de ulike kategoriene i et panel. Valg av en spørring, medfører at en popup[6] vises med informasjon om spørringen samt hvilke parametre som må settes. Disse popup-vinduene vil vises og presenteres sammen med de tilhørende spørringene i neste kapittel.

[5]Et element som vises når markøren interagerer med en knapp, ikon eller link [81]
[6]Et vindu som dukker opp og dekker over det tidligere viste vinduet



Figur 3.3: Tooltip bestående av linker til spørringer under kategorien *Positioning*

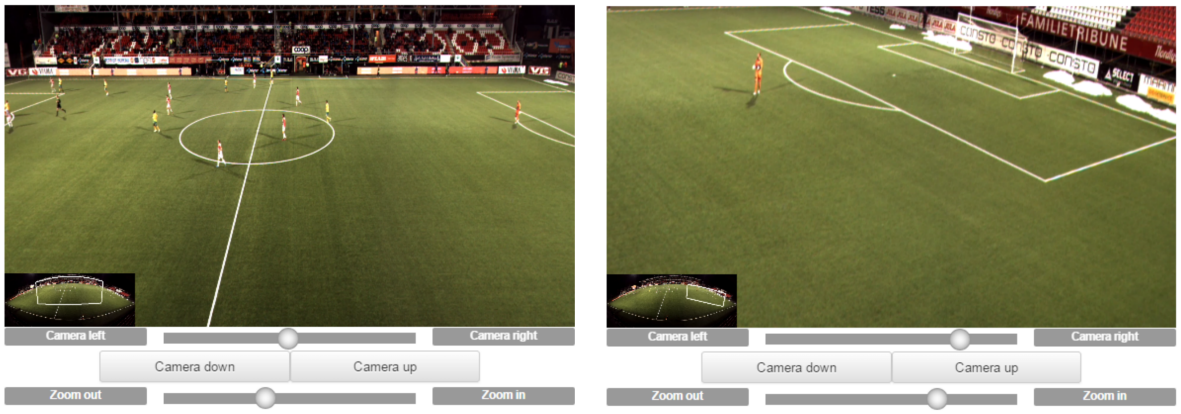
3.5 Virtuell kamerastyring

Konseptet om Bagadus' virtuelle kamera presenteres i seksjon 2.2.1. Videosystemet bruker det sylindriske panoramabildet til å generere en *viewer* som gjør det mulig for brukere å selv bestemme hvilke områder av banen som skal vises i videoavspilleren. For å kunne strømme video av kampene i Bagadus-databasen, behøver en eventuell klientside kun støtte for WebSocket og HTML5 Video.

Analyseverktøyet inneholder funksjonalitet for flytting og zooming av kameraet når situasjoner skal avspilles. Idet kampanalysesiden lastes, startes en WebSocket og en ControlSocket. WebSocketen håndterer kommandoer tilknyttet zooming og panorering av panoramabildet, mens ControlSocketen håndterer argumenter som bestemmer hvilken del av kampen som skal strømmes. Operasjonene tilknyttet flytting og zooming av kamera utføres når brukerne interagerer med *sliderne* og knappene under videospilleren, mens tidspunktene velges via listen som inneholder spørringsresultatene (se høyre panel i figur 3.3). Verdiene for zooming og flytting av kamera horisontalt og vertikalt har rekkevidder fra 0 til 100 (se figur 3.4). Listing 3.7 viser et utsnitt av operasjonene som utføres når videostrømmer skal spilles.

Listing 3.7: WebSocket og ControlSocket i analyseverktøyet

```
$scope.websocket = new WebSocket('ws://<ip>:<port>/websocket/');
$scope.controlsocket = new WebSocket('ws://<ip>:<port>/');
```



x=50, y=30, f=40 x=80, y=30, f=70

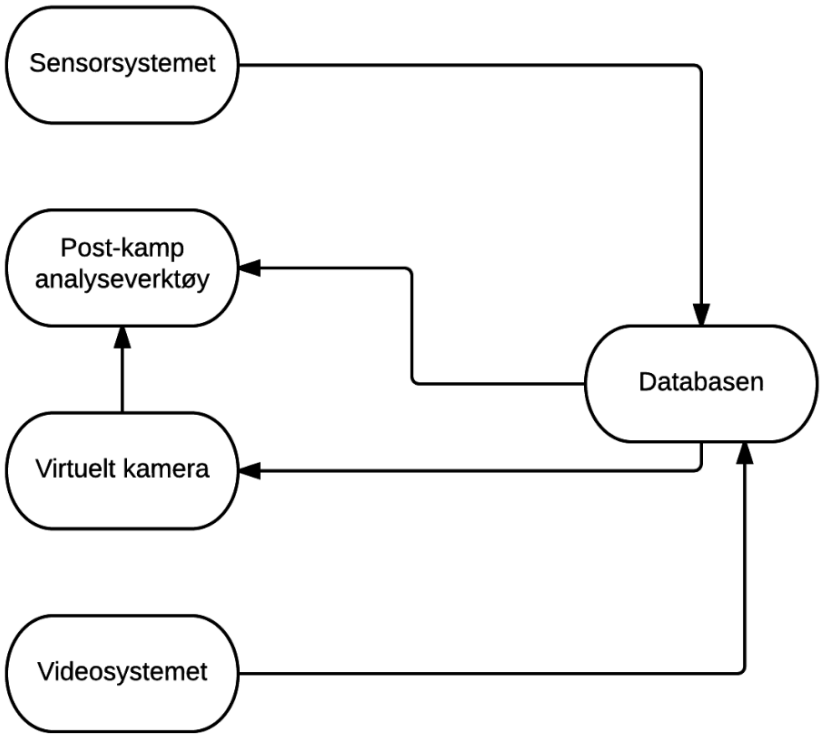
Figur 3.4: Analyseverktøyets virtuelle kamera

```
$scope.moveCameraUp = function() {
  if($scope.cameraY == 0) {
    return
  }
  $scope.cameraY = $scope.cameraY - 10;
  $scope.websocket.send("x="+$scope.cameraX+";y="+$scope.cameraY+";");
};

$scope.playVideo = function() {
  var tmp = JSON.stringify({'command':'custom_event',
    'start':$scope.currentVideoClip.start,
    'stop': $scope.currentVideoClip.end, 'play': true});
  console.log(tmp);
  $scope.controlsocket.send(tmp);

};
```

Det virtuelle kameraet som er implementert i prototypen, er basert på den manuelle avspillingsfunksjonen som presenteres i seksjon 2.2.1. Videosystemet støtter også en automatisk modus hvor flytting av kamera er basert på posisjonene til den eller de spillerne som *trackes*. Integrering av denne funksjonen i prototypen vil imidlertid ikke skje under nåværende iterasjon, men det vil utgjøre en viktig del av den fremtidige videreutviklingen [22].

3.6 Arkitektur

Figur 3.5 viser en overordnet skisse av hvordan analyseverktøyet kommuniserer med Bagadus' eksisterende komponenter og systemer. Når en spørring har blitt konstruert, eksekveres den mot databasen. Denne databasen inneholder informasjon som er produsert av sensorsystemet og videosystemet. Tidspunktene som returneres, sendes deretter som argumenter til ControlSocketen som håndterer det virtuelle kameraet, og analyseverktøyets videospiller viser situasjonen.



Figur 3.5: Komplett arkitektur

Analyseverktøyet består av en webserver, en applikasjonsserver og en databaseserver. Webserveren håndterer operasjoner tilknyttet brukergrensesnittet og kommunikasjon med backend-delen. Backend-delen utgjør applikasjonsserveren og håndterer logikk og kommunikasjon med databasen. Figur 3.6 gir et grovt blikk av de ulike komponentene i analyseverktøyet og visualiserer operasjonene som utføres når spørringer konstrueres og eksekveres. Et eksempel på en spørring kan være å finne alle situasjoner i kamp Ms første omgang, hvor spiller X løper fortere 8 meter per sekund innenfor venstre 16-metersboks på Alfheim stadion (beskrives i seksjon 4.3.4). Operasjonene som utføres når denne spørringen skal eksekveres, inkluderer følgende:

1. Navnet på kamptabellen hentes idet analysesiden for den gitte kampen lastes og lagres i en AngularJS-variabel. Videre defineres valg av spiller, løpshastighet, omgang

og område i popup-vinduet for spørringen. Verdiene for disse parametrene lagres midlertidig i variabler som også er deklarert i AngularJS-kode. For at spørringen skal ta stilling til valg av omgang, inkluderes en siste variabel i parameterlisten, kampens starttidspunkt, som hentes idet kampanalysesiden lastes. Følgende spørring henter dette tidspunktet:

```
SELECT min(time)
FROM Match m;
```

2. Idet søkeknappen trykkes, kalles en AngularJS-metode som prosesserer parametrene og sender dem til den aktuelle metoden i Java-kontrolleren (*MatchController.java*).

3. Den aktuelle metoden i kontrolleren mottar parametrene i selve URL-stien, og argumentene er derfor definert som *sti-variabler* [70]. Parametrene sendes deretter videre til den aktuelle metoden i tjenesten (i *MatchService.java*) som konstruerer spørringen.

4. Tjenesten mottar argumentene, prosesserer dem og inkluderer dem som betingelser i den konstruerte spørringen. Dette resulterer i en tekststreng som inneholder følgende databasekode:

```
SELECT player, time-10 sec, time+10 sec
FROM Match m
WHERE player = X AND
      velocity >= 8 AND
      pos_x BETWEEN 0 and 16,5 AND
      pos_y BETWEEN 20 AND 50 AND
      time < (start_time + 55 minutes)
```

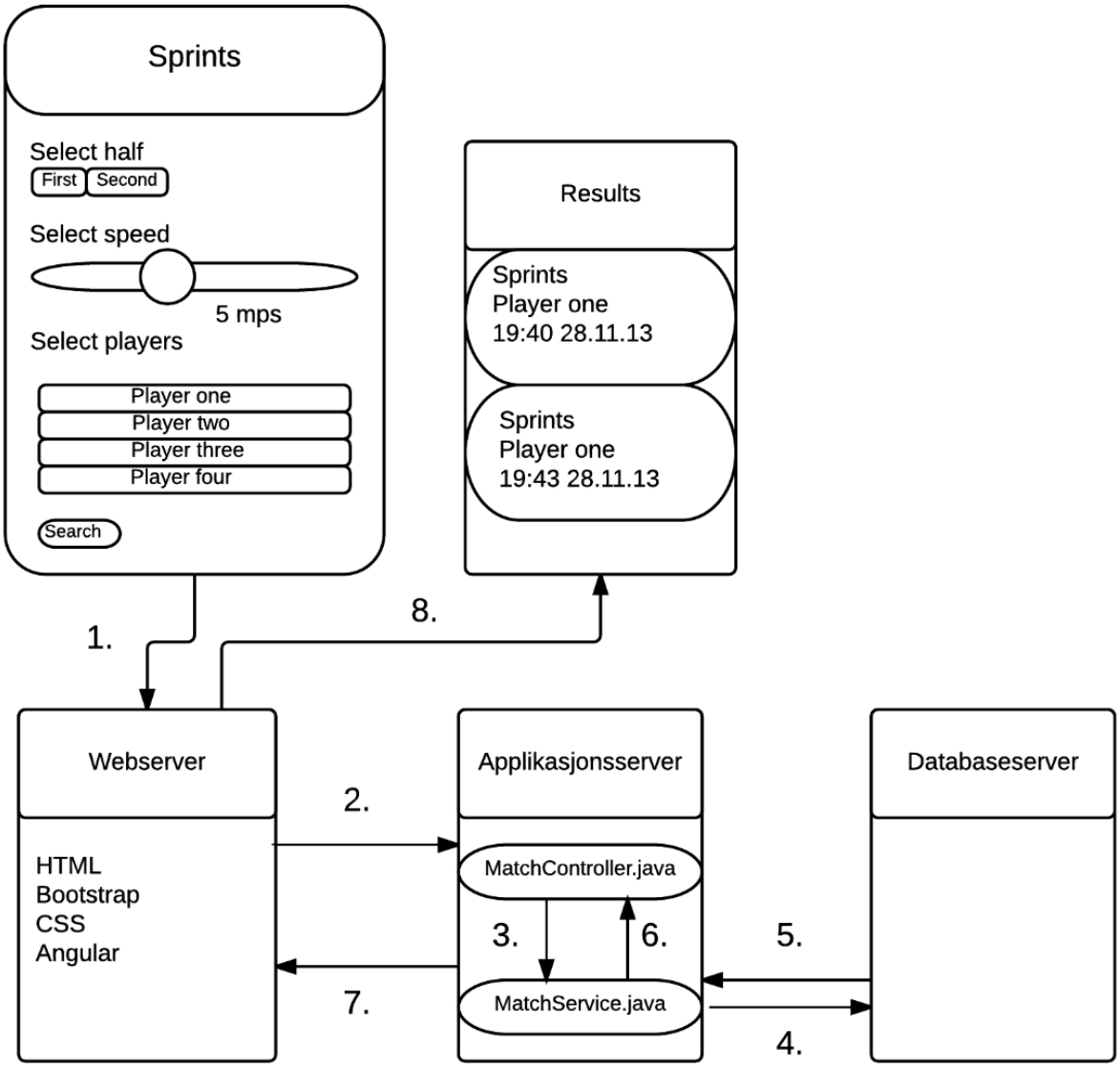
   Spørringen spesifiserer at spiller X må oppnå en løpshastighet på minst 8 meter per sekund innenfor koordinatene som utgjør venstre 16-metersboks på banen. Den siste linjen i koden bestemmer at kampsituasjonen må oppstå mellom kampens starttidspunkt og 55 minutter senere. Bakgrunnen for denne metoden for å skille omganger på, beskrives i seksjon 4.2. Til slutt sendes spørringen til databaseserveren.

5. Når spørringen har blitt eksekvert, mottar den aktuelle metoden i tjenesten resultatet i form av et *ResultSet*-objekt [50] som inneholder rader med spilleridentifikasjoner og tidsintervaller. Dette objektet returneres videre til kontrolleren.

6. Java-kontrolleren mottar *ResultSet*-objektet og konverterer innholdet til JSON-objekter [34]. Disse objektene returneres deretter til et REST-API [74].

7. Webserveren henter JSON-objektene fra REST-APIet (via AngularJS) og lagrer dem i en resultatliste.

8. Resultatet visualiseres til slutt som en spilleliste i brukergrensesnittet. Ved å dobbelklikke på et element i denne spillelisten, sendes situasjonens start- og sluttidspunkt til ControlSocketen som håndterer det virtuelle kameraet. Systemets videospiller viser deretter situasjonen.



Figur 3.6: Analyseverktøyets arkitektur

3.7 Sammendrag

I dette kapittelet har en prototype av et post-kamp analyseverktøy blitt presentert. Dette verktøyet integrerer Bagadus' sensor- og videosystem og inneholder konstruerte SQL-spørringer som finner statistikk og video av kampsituasjoner. Bakgrunnen for implementasjon har rot i at Bagadus mangler brukergrensesnitt. Uthenting av informasjon fra Bagadus-databasen har derfor tidligere vært en funksjonalitet begrenset til personer med teknisk bakgrunn. I neste kapittel presenteres spørringer som finner tidspunkt i kamper hvor bestemte hendelser oppstår. Disse vil optimaliseres for å tilby brukerne av systemet en hurtig og effektiv måte å utføre kampanalyse på.

Kapittel 4

Ekstrahering av kampsituasjoner

I dette kapittelet presenteres ferdigkonstruerte SQL-spørringer som finner tidspunkt i kamper hvor spillere posisjonerer eller beveger seg på bestemte måter. Disse spørringene utgjør en viktig del av analyseverktøyet som presenteres i denne oppgaven og vil være fleksible i den forstand at betingelsene ikke er konstante, men i stedet basert på parametre som brukerne definerer i brukergrensesnittet.

Flere av seksjonene presenterer ulike løsninger av samme spørring, og disse har blitt sammenlignet med kriterier som spesifiseres i seksjon 4.1. Videre presenteres hjelpemetoder- og variabler som er implementert. Deretter presenteres hver enkelt spørring med beskrivelse av bakgrunn for spørringen, hvordan den har blitt implementert, hvordan den har blitt testet og hvordan parametrene defineres i brukergrensesnittet. Til slutt følger en kortfattet beskrivelse av mulige utvidelser.

4.1 Testing

Responstid er et av de ikke-funksjonelle kravene som stilles til analysesystemet. Det er derfor essensielt at spørringer som finner statistikk og video av kampsituasjoner, er optimalisert til å returnere resultat på kortest mulig tid. I tilfeller hvor konsepter til spørringer har ulike fremgangsmåter, har ulike løsninger blitt konstruert. Kjøretidene til disse har deretter blitt sammenlignet, hvorav den mest tidseffektive løsningen har blitt implementert i analyseverktøyet.

På forhånd ble det gjort et litteraturstudium for å finne liknende problemstillinger og undersøkelser. Blant fagrelaterte funn var [12], [77], [8] og [36]. Disse fremstilte imidlertid andre formål enn det som presenteres i denne oppgaven. På bakgrunn av dette ble det formulert en rekke mål og kriterier utfra det overordnede målet; å implementere løsninger

som hurtig genererer statistikk og videosammendrag fra fotballkamper. Praktiske spørsmål tilknyttet tid, samt momenter med påvirkningskraft på tidsbruk har blitt tatt hensyn til, og løsningene har blitt eksekvert med utgangspunkt i følgende kriterier:

- **Samtlige spørringer bør eksekveres på samme datamaskin.** Kvaliteten på hardware har en innvirkning på hvor effektivt programvare lar seg kjøre. Ulike datamaskiner har ulik ytelse, og ved lav prosessorkraft utfører systemer operasjoner tregere. Hardware kan derfor påvirke hvor hurtig en spørring returnerer resultat, og dette medfører at identiske spørringer kan ha ulike kjøretider på ulike datamaskiner.
- **Samtlige spørringer bør eksekveres mot samme database og tabell.** Mengden informasjon som finnes i en databasetabell, har en innvirkning på kjøretid. Jo flere rader et skjema inneholder, jo mer informasjon må en spørring prosessere.
- **Samtlige løsninger av samme spørring bør eksekveres med like parametre og betingelser.**
- **Samtlige løsninger av samme spørring bør eksekveres i samme omgang og i en vekselvis rekkefølge.** Dersom forutsetninger for kjøring endres under testene, vil dette påvirke kjøretidene. Ved å eksekvere dem i vekselvis rekkefølge vil imidlertid eventuelle systematiske feil ikke påvirke i like stor grad.
- **Parameter-verdiene i spørringene bør ha store rekkevidder.** Når flere skjemaer i en spørring skal kombineres, vil kompleksiteten og kjøretidene øke med mengden informasjon som finnes i hvert skjema. Dette demonstreres med et eksempel i seksjon 4.3.3.
- **Samtlige spørringer bør eksekveres 10 ganger med kommandoen *explain analyze*** [60]. Denne kommandoen vil returnere kjøretiden.
- **Før spørringer eksekveres, bør alle kjørende programmer som krever mye minne, avsluttes**. Slike programmer inkluderer blant annet Spotify, Chrome og Skype. Bakgrunnen for dette er at antall prosesser som kjøres på en gang, har en innvirking på datamaskinens ytelse og systemers evne til å utføre oppgaver effektivt. Kjørende programmer som krever mye minne og komputeringskraft kan derfor ha en innvirkning på hvor hurtig spørringer returnerer resultater.

Spørringene har blitt eksekvert mot et utsnitt av Bagadus-databasen. Denne har inneholdt totalt 493.663 rader med sensorinformasjon fra første omgang av europaligakampen (Europe League) mellom Tromsø og Tottenham som ble spilt i 2013.

PostgreSQL har, i likhet med andre databasespråk, innebygde algoritmer og metoder for optimalisering av spørringer. *The query optimizer* tar en gitt spørring som inndata og

utfører operasjoner som genererer en eksekveringsplan. Denne planen konstrueres med mål om å returnere resultat på en effektiv måte. Antall mulige planer økes parallelt med størrelsen på en spørring og antall operasjoner som utføres. Hvorvidt en løsning er mer effektiv en annen, er derfor avhengig av hvordan løsningene er strukturert og hvordan optimisereren tolker inndataen [11]. PostgreSQL-systemet er komplekst, og det florerer av teorier, i fora som *Stackoverflow.com* og *PostgreSQL.com*, om hvorfor noen fremgangsmåter for konstruksjon av spørringer er bedre enn andre. Målet med testene som utføres i dette kapittelet, er ikke å produsere generaliserbare resultater. I et slikt tilfelle burde i så fall antallet eksekveringer økes, og eventuelle konklusjoner ville ha forutsatt kunnskap om alle operasjoner og algoritmer som PostgreSQL utfører. I stedet er formålet med testene å finne løsninger som effektivt genererer statistikk og videosammendrag basert på fysiske spillerdata. På bakgrunn av dette formålet, samt de forberedelsene og tiltakene som gjøres i forkant av testene, har 10 blitt vurdert som et tilstrekkelig antall eksekveringer per løsning for å oppnå et grunnlag for sammenligning. Når testresultatene beskrives, vil teorier om utfallene diskuteres. Det vil imidlertid ikke trekkes konklusjoner ettersom dette igjen vil kreve kunnskap om alle operasjoner og algoritmer som PostgreSQL utfører.

Når samtlige løsninger av samme spørring har blitt kjørt 10 ganger, utregnes gjennomsnitt og median av kjøretidene. Disse verdiene har deretter dannet grunnlaget for avgjørelsen om hvilken løsning som inkluderes i analyseverktøyet. Ettersom løsningene eksekveres under like forutsetninger vil alltid den raskeste, uavhengig av margin, være den som implementeres. Dersom marginen er lav, er det imidlertid ingen garanti for at den foretrukne løsningen er den raskeste i alle tilfeller. Et slikt tilfelle vil uansett tyde på at løsningene er tilnærmet like med henblikk på kompleksitet.

4.2 Hjelpemetoder- og variabler

Avrunding av tid

Kamptabellene i Bagadus-databasen inneholder informasjon om spillernes posisjoner på banen, og denne informasjon lagres 20 ganger i sekundet under kampene. Ettersom en spiller ikke kan forlytte seg særlig langt på 1/20-dels sekund, er det realistisk å anta at det vil være små forskjeller mellom en rad og den neste i tabellene. En spørring som ber om tidspunkt i en kamp hvor en spiller posisjonerer seg i et bestemt område, vil returnere 200 tidspunkt dersom spilleren befinner seg der i 10 sekunder. Det har derfor vært nødvendig med en funksjon som samler disse tidspunktene til èn situasjon. I den anledning ble det konstruert en SQL-funksjon [59] som avrunder tidspunkt (se

listing 4.1). Denne funksjonen tar to parametre; et tidspunkt og en tallverdi. Tallverdien bestemmer hvordan tidspunktet skal avrundes (10 = nærmeste tiende sekund, 100 = nærmeste hundrede sekund). Ved å inkludere *distinct* [63] i projeksjonene[1] i spørringene vil dermed alle situasjoner som avrundes til samme tidspunkt, samles i en rad.

Listing 4.1: SQL-funksjon som avrunder tidspunkt

```
create or replace function round_timestamp(
timestamp, integer) returns
timestamp as $$
select 'epoch'::timestamp + '1 second'::interval
* ($2 * round(date_part('epoch', $1) / $2));
$$ language sql immutable;
```

Funksjonen bruker PostgreSQL sin innebygde metode *date_part()* til å finne tidspunktets *epoch* (antall sekunder siden 00:00 1.1.1970), og denne verdien divideres på parameteret som definerer hvordan tidspunktet skal avrundes. Videre avrundes resultatet til nærmeste heltall for deretter å multipliseres med parameteret igjen [57].

Start- og sluttidspunkt

Når kampsituasjoner skal analyseres, en det hensiktsmessig at video inneholder forhåndsdefinerte start- og sluttidspunkt som viser hva som ledet til at situasjonen oppsto og hva den endte i. På bakgrunn av dette, vil spørringene som presenteres i dette kapittelet bestå av projeksjoner som inneholder to kolonner med tider som definerer et tidsintervall. Den første kolonnen vil være det avrundede tidspunktet subtrahert med et bestemt antall sekunder, og den andre vil være tidspunktet addert med et bestemt antall sekunder.

Samtlige tidspunkt som spørringene finner, rundes til nærmeste tiende sekund. Dersom noen tidsintervaller overlapper, slås de sammen i den aktuelle kontrollermetoden. Dette gjør at resultatene som returneres vil inneholde hele situasjoner.

Omganger

Tabellene i Bagadus-databasen skiller ikke mellom omgangene i kampene. I stedet lagres all informasjon i samme kamptabell med spilleridentifikasjon og tidspunkt som nøkkel i hvert tuppel[2]. Ettersom det kan eksistere scenarioer hvor trenere og andre brukere kun er

[1] SQL-kode som spesifiserer hvilke kolonner som skal returneres av en spørring
[2] Rad i en databasetabell

interessert i en av omgangene, har det vært nødvendig å finne en måte å skille omgangene.

En omgang har ordinært en varighet på 45 minutter. Det er imidlertid vanlig at det legges til minutter dersom det har vært flere stopp i spillet forårsaket av spillerbytter, skader eller liknende. Denne tilleggstiden er som regel fra ett til fire minutter. Mellom hver omgang har lagene 15 minutter pause. På bakgrunn av denne informasjonen har vi valgt å la første omgang vare fra første spillesekund til 55 minutter senere. Andre omgang er satt til å vare fra 60 minutter etter første spillesekund til siste spillesekund.

Kampens starttidspunkt hentes idet kampanalysesiden lastes. Dette tidspunktet lagres i en AngularJS-variabel kalt *start_time*, og inkluderes i parameterlisten som sendes når spørringer skal konstrueres. Listing 4.2 viser hvordan argumentet *start_time* brukes til å definere omgang. Eksempelet spesifiserer at kampsituasjonen skal oppstå mellom kampstart og 55 minutter senere, altså i første omgang.

Listing 4.2: Eksempel på spørring som finner alle situasjoner hvor spiller X løper fortere enn 8 meter per sekund i første omgang

```
SELECT player, time
FROM match
WHERE player = X AND
velocity > 8 AND
time < (start_time + '55' min);
```

Hvor er spillerne i første omgang?

Ved eksekvering av spørringer med parametre som angår bestemte områder på banen, er det essensielt at brukerne vet hvilken side hjemmelaget angriper fra i hver omgang. I den anledning ble det konstruert en spørring som eksekveres idet kampanalysesiden lastes. Avspark tas av to spillere som befinner seg i sentrum av midtsirkelen på banen. Resten av spillerne må posisjonere seg utenfor denne sonen, på egen halvdel. Spørringen er basert på spillernes sensorkoordinater i første spillesekund. Dersom spillerne befinner seg på venstre banehalvdel (fra kameraenes perspektiv), returneres tekststrengen *left*. Tekststrengen *right* returneres dersom spillerne befinner seg på høyre banehalvdel. Denne strengen lagres i en AngularJS-variabel kalt *sideInFirstHalf*. Informasjon om hvilken side laget angriper fra i hver omgang, presenteres for brukerne når parametre for spørringer skal defineres.

4.3 Spørringer

4.3.1 Spiller i område

Bakgrunn

Formasjoner og taktikker er basert på hvor og hvordan bestemte spillere skal posisjonere og bevege seg på banen. For eksempel inkluderer en kantspillers plikt ofte produsering av innlegg fra sidene, mens en dyptliggende playmaker bør stasjonere seg sentralt på midtbanen og styre spillet. Ulike spillere i ulike posisjoner har ansvar knyttet til ulike soner på banen, og gjennomføring av kampplaner avhenger av at spillerne utfører sine tildelte plikter innenfor de bestemte sonene.

Mortensen presenterer i [41], tre eksempelspørringer mot sensordatabasen. En av dem, *In opponents 18-yard box*, finner alle situasjoner hvor en spesifikk spiller befinner seg i motstanders 16-metersboks. Spørringen som presenteres i det påfølgende, er en utvidelse av denne eksempelspørringen.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen 20 ganger i sekundet. Dette danner bakgrunnen for spørringen *Spiller i område* som finner alle situasjoner hvor en spesifikk spiller befinner seg innenfor et avgrenset område på banen.

Frontend

Spiller i område skal fungere som et virkemiddel i analysering av spilleres posisjoneringsmønstre. I kampanalysesiden er den derfor representert i kategorien *Positioning* (posisjonering). Popup-vinduet som vises i figur 4.1 inkluderer følgende elementer for definering av parametre:

- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Spiller(e)**: En liste med spilleridentifikasjoner som støtter markering av flere elementer
- **Område**: Markering av felt på en opptegnet fotballbane

Figur 4.1: Popup for setting av parametre for spørringen *Spiller i område*

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Spiller i område* til databaseserveren, tar åtte argumenter:

- Navnet på kamptabellen
- En liste med spillere
- Fire koordinater som definerer det avgrensede området
- En tekststreng som definerer hvilken omgang som skal analyseres
- Kampens starttidspunkt

Projeksjonene i løsningene som presenteres under, består av tre kolonner i form av spilleridentifikasjon og to tidspunkt som definerer et tidsintervall (se listing 4.3 og 4.4). Tidsintervallet brukes for å hente video, mens spilleridentifikasjonen inkluderes for å opplyse bruker om hvilken spiller hver situasjon er knyttet til.

For å finne alle situasjoner hvor en bestemt spiller befinner seg innenfor et avgrenset område på banen, sammenlignes parametrene med følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Spørringene som vises i listing 4.3 og 4.4 presenterer to løsninger som returnerer samme resultat. Data hentes fra ett skjema, kamptabellen, og spørringene finner alle situasjoner hvor en, og ikke samtlige av de valgte spillerne oppfyller kriteriene. Tidsintervallet som defineres har en lengde på 20 sekunder, og dersom returnerte intervaller overlapper, slås de sammen i den aktuelle metoden i kontrolleren.

Koden i listing 4.3 bruker kommandoen *OR* for å ta stilling til flere spillere i parameterlisten. Denne kommandoen gjør at spørringen vil finne alle situasjoner hvor en av spillerne oppfyller resten av kriteriene i *where*-klausulen [90]. Eksempelet under finner alle situasjoner som oppstår i andre omgang, og parametrene er markert rødt i koden.

Listing 4.3: Eksempel 1 - *Spiller i område* i pseudokode

```
SELECT player, round_time(time,10)-10 sec, round_time(time,10)+10 sec
FROM  Match
WHERE (player = player1 OR player = player2) AND
pos_x between X1 AND X2 AND
pos_y between Y1 AND Y2 AND
time > (start_time + 60 min)
```

PostgreSQL-kommandoen *UNION* [58] brukes for å kombinere spørringer og er et alternativ til kommandoen *OR*. Eksempelet i listing 4.4 inneholder én spørring per spiller i parameterlisten, og disse kombineres ved å inkludere *UNION* mellom dem. Delspørringene er dermed identiske med unntak av betingelsen tilknyttet spiller. En forutsetning for bruk, er at delene har et likt antall kolonner og liknende datatyper i projeksjonene. Når hele spørringen eksekveres, vil den for hver iterasjon returnere rader som oppfyller kravene i en av delspørringene. Parametrene er markert rødt i koden.

Listing 4.4: Eksempel 2 - *Spiller i område* i pseudokode

```
SELECT player, round_time(time,10)-10 sec, round_time(time,10)+10 sec
FROM  Match
WHERE player = player1 AND
pos_x between X1 AND X2 AND
```

```
pos_y between Y1 AND Y2 AND
time > (start_time + 60 min)
UNION
SELECT player, round_time(time,10)-10 sec, round_time(time,10)+10 sec
FROM  Match
WHERE player = player2 AND
pos_x between X1 AND X2 AND
pos_y between Y1 AND Y2 AND
time > (start_time + 60 min)
```

Testing

For å sammenligne løsningene (se Vedlegg A for fullstendig kode), har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze*. Denne kommandoen vil skrive ut kjøretiden til hver spørring i millisekunder. Parametrene er predefinerte og presenteres under:

- **Kamp**: Tromsø-Tottenham
- **Spillere**: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 og 14
- **Område**: 0.525 <X <17,0625, 14.8128 <Y <55.6495
- **Omgang**: Første

Derek Dieter sammenligner i [17], bruk av kommandoene *OR* og *UNION* i SQL Server [40]. Han refererer til uttrykkene *seek* og *scan* og skriver at *OR* generelt er mindre effektiv. Bakgrunnen for dette er at denne kommandoen medfører at hele indekser og tabeller skannes oftere. I kontrast bruker ofte *UNION* tre-strukturen i indeksene for å hente like elementer (*seek*). Dinesh Asanka påpeker imidlertid i sitt innlegg [6] at forskjellene mellom de to kommandoene avhenger av formålet med og strukturen i spørringen som eksekveres.

De ovennevnte innleggene omhandler SQL Server og ikke PostgreSQL. Det er derfor diskutabelt hvorvidt synspunktene kan gjelde for spørringen som har blitt presentert i denne seksjonen. Resultatene viser at løsningen i listing 4.4 bruker omlag 367 millisekunder i gjennomsnitt på å returnere resultat (se tabell 4.1), mer enn dobbelt så fort som kjøretidene til løsningen i 4.3. I tillegg medfører bruk av *UNION* langt mer kode (en delspørring per betingelse tilknyttet spiller). På bakgrunn av kjøretidene, samt det potensielle omfanget av kode, vil *UNION* ikke benyttes ved flere anledninger i denne oppgaven. Spørringen i listing 4.3 er derfor implementert i analyseverktøyet.

Tabell 4.1: Sammenligning av spørringene i listing 4.3, og 4.4

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.3)	145,671 ms, 161,049 ms 165,005 ms, 165,237 ms 167,100 ms, 170,393 ms 170,787 ms, 181,834 ms 184,226 ms, 185,804 ms	169,7106 ms	168,7465 ms
Eksempel 2 (4.4)	359,098 ms, 362,483 ms 362,965 ms, 364,058 ms 364,198 ms, 366,456 ms 367,326 ms, 368,254 ms 378,956 ms, 381,740 ms	367,5534 ms	365,327 ms

4.3.2 Avstand mellom to spillere

Bakgrunn

Three metres distance from another player er den andre av tre eksempelspørringer som presenteres i [41] og finner alle situasjoner hvor en spesifikk spiller befinner seg mindre enn tre meter fra en annen spiller. Denne spørringen introduserer muligheter for ekstrahering av flere ulike kampsituasjoner i ulike ledd på banen. I en forsvarende situasjon er det primært forsvarsspillernes oppgave å minimalisere antall målsjanser imot. Motstanderne må dekkes, og forsvarsleddet må opprettholde et kollektivt samarbeid. En viktig forutsetning for et godt kollektivt samarbeid, er at forsvarerne posisjonerer seg riktig i forhold til motspillerne og hverandre. Dersom to spillere holder for kort avstand til hverandre, åpnes det større rom andre steder for motstanderne til å spille seg gjennom.

Avstand mellom spillere kan også være et avgjørende element i et angrepsspill. Under 2013/2014-sesongen i Premier League, uttrykte en av Manchester Uniteds angrepsspillere, Robin van Persie, misnøye over sine egne lagkamerater. Han sa at medspillerne hans ofte løp i rommene som han ønsket å bevege seg i. Et viktig element i en angripende situasjon, er bevegelse. For å unngå at spillet stopper opp og blir statisk, må ulike spillere bevege seg i ulike rom [78].

Spørringen som presenteres i [41] er basert på tre parametre; to spillere og avstanden mellom dem. Resultatet som returneres vil være situasjoner som oppstår i samtlige baneledd.

En realistisk antakelse er imidlertid at en trener kun er interessert i avstanden mellom to spillere i et bestemt område på banen. Ved undersøkelse av påstanden til Robin van Persie om at medspillerne hans løper i veien for ham, kunne det vært aktuelt å avgrense området til den siste tredjedelen av banen ettersom han er en angrepsspiller.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen. Dette danner bakgrunnen for spørringen *Avstand mellom to spillere* som finner alle situasjoner hvor to spesifikke spillere holder mer eller mindre enn en viss avstand til hverandre innenfor et avgrenset område på banen.

Frontend

Spørringen *Avstand mellom to spillere* finner situasjoner som angår hvordan spillere posisjonerer seg i forhold til hverandre i gitte baneledd. I kampanalysesiden er den derfor representert i kategorien *Positioning* (posisjonering). Popup-vinduet som vises i figur 4.2 inkluderer følgende elementer for definering av parametrene:

- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Avstand**: En *slider* med rekkevidde fra 1 til 10 meter og en knapp som bestemmer om avstanden skal være større eller mindre
- **Spillere**: To *select*-bokser [85] med spilleridentifikasjoner
- **Område**: Markering av felt på en opptegnet fotballbane

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Avstand mellom to spillere* til databaseserveren, tar 11 argumenter:

- Navnet på kamptabellen
- Avstanden mellom to spillere
- To spillere
- Fire koordinater som definerer det avgrensede området

Figur 4.2: Popup for setting av parametre for spørringen *Avstand mellom to spillere*

- En tekststreng som definerer hvilken omgang som skal analyseres
- En tekststreng som bestemmer hvordan avstanden skal sammenlignes (<>)
- Kampens starttidspunkt

Projeksjonene i løsningene som presenteres under, består av fire kolonner som inkluderer to spillere og to tidspunkt. Tidspunktene brukes for å hente video, mens spilleridentifikasjonene inkluderes for å opplyse bruker om hvilke spillere hver situasjon er knyttet til.

For å finne avstanden mellom to geometriske punkt, har PostgreSQL sin innebygde funksjon *circle* [61], blitt benyttet. Funksjonen tar to parametre, ett punkt og en tallverdi, og disse brukes til å definere en sirkel med punktet som sentrum og tallverdien som radius. Ved å benytte kommandoene *IN* eller *NOT IN*, kan man deretter undersøke om et annet punkt enten befinner seg i eller utenfor sirkelen.

For å finne alle situasjoner hvor to spesifikke spillere holder mer eller mindre enn en viss avstand fra hverandre, sammenligner spørringen de brukerdefinerte parametrene med

følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

I SQL er det imidlertid ikke mulig å sammenligne to verdier av samme attributt fra én tabell. For å sammenligne koordinatene til to spillere, må derfor informasjonen om dem hentes fra to separate tabeller. I spørringen som vises i listing 4.5, er samme kamptabell definert to ganger, og en *where*-klausul brukes for å kombinere dem og filtrere ut radene som returneres. Denne fremgangsmåten for kombinering av tabeller kalles *implicit join*. De initiale tidspunktene som spørringen finner, avrundes til nærmeste tiende sekund, og den resulterende tiden brukes til å definere et intervall på 20 sekunder. I tillegg vil returnerte intervaller som overlapper, slås sammen i den aktuelle metoden i kontrolleren. Parametrene er markert rødt i koden.

Listing 4.5: Eksempel 1 - *Avstand mellom to spillere* i pseudokode

```
SELECT one.player, two.player, round_time(one.time,10)-10 sec,
       round_time(one.time,10)+10 sec
FROM Match one, Match two
WHERE one.player != two.player AND
one.player = player1 AND
two.player = player2 AND
one.time = two.time AND
one.pos_x between X1 AND X2 AND
one.pos_y between Y1 AND Y2 AND
two.pos_x between X1 AND X2 AND
two.pos_y between Y1 AND Y2 AND
one.pos IN circle(two.pos,distance) AND
one.time > start_time + 60 min
```

Spørringen som vises i listing 4.6 oppretter to skjemaer, ett for hver spiller, via kommandoen *with* [64]. Slike skjemaer kalles *Common Table Expressions* og beskrives i [77] som midlertidige *views* eller tabeller. De lagres midlertidig i minnet og kan kun aksesseres av den påfølgende spørringen. Når spørringen har blitt eksekvert, fjernes *viewet*. idéen med denne fremgangsmåten er å begrense mengden informasjon som må sammenlignes og prosesseres i det ytre leddet av koden, et aspekt som beskrives i [37]. Konseptet om å begrense informasjon vises også i listing 4.7, men i stedet defineres skjemaene som subspørringer.

Via *where*-klausuler kombineres skjemaene på tidspunkt og den gitte avstanden mellom spillerne.

Listing 4.6: Eksempel 2 - *Avstand mellom to spillere* i pseudokode

```
WITH player_one AS
     (SELECT player, time, pos
      FROM  Match
      WHERE player =  player1 AND  ...),
      player_two AS
     (SELECT player, time, pos
      FROM  Match
      WHERE player =  player2 AND  ...)
SELECT player_one.player, player_two.player,
       round_time(player_one.time,10)-10 sec,
       round_time(player_one.time,10)+10 sec
FROM player_one one, player_two two
WHERE player_one.time = player_two.time AND
one.pos IN circle(two.pos,distance)
```

Listing 4.7: Eksempel 3 - *Avstand mellom to spillere* i pseudokode

```
SELECT player_one.player, player_two.player,
       round_time(player_one.time,10)-10 sec,
       round_time(player_one.time,10)+10 sec
FROM
      (SELECT player, time, pos
       FROM Match
       WHERE player =  player1 AND  ...) player_one,
      (SELECT player, time, pos
       FROM Match
       WHERE player =  player2 AND  ...) player_two
WHERE player_one.time = player_two.time AND
      one.pos IN circle(two.pos,distance)
```

Testing

For å sammenligne løsningene (se Vedlegg B for fullstendig kode), har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze*. Parametrene er predefinerte og presenteres under:

- **Kamp**: Tromsø-Tottenham
- **Spiller 1**: 1
- **Spiller 2**: 4
- **Avstand**: <5 meter
- **Område**: Hele banen
- **Omgang**: Første

Spørringen i listing 4.6 (som bruker *with* for å opprette skjemaer) bruker omlag 400 sekunder på å returnere resultat (se tabell 4.2). Sammenlignet med de to andre løsningene som har gjennomsnittlige kjøretider på rundt 290 millisekunder, er den derfor svært ineffektiv. På bakgrunn av dette, ble den kun eksekvert to ganger. Ellen Munthe-Kaas lister i [42] opp faktorer som kan øke kostnaden når spørringer skal eksekveres. Blant faktorene finnes bruk av indekser som ikke er lagret i minnet. En grunn til at oppretting av skjemaer via kommandoen *with*, i noen tilfeller kan være treg, er at skjemaene lagres midlertidig i minnet. Dersom attributtene i skjemaene ikke er indeksert, vil oppslag på et slikt attributt i den ytre spørringen, kreve at tabellen itereres gjennom flere ganger, og dette kan øke kompleksiteten betraktelig.

Tabell 4.2: Sammenligning av spørringene i listing 4.5, 4.6 og 4.7

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.5)	270,783 ms, 273,981 ms 276,901 ms, 277,091 ms 286,439 ms, 289,113 ms 289,822 ms, 291,974 ms 303,951 ms, 309,616 ms	286,9671 ms	287,776 ms
Eksempel 2 (4.6)	403601,825 ms 396935,327 ms	400268,576 ms	400268,576 ms
Eksempel 3 (4.7)	269,767 ms, 270,709 ms 272,940 ms, 284,506 ms 285,196 ms, 289,175 ms 290,852 ms, 297,088 ms 298,214 ms, 314,122 ms	287,2569 ms	287,1855 ms

Løsningene i listing 4.5 og 4.7 bruker henholdsvis kamptabellen og subspørringer til å opprette skjemaer. Testresultatene viser at mindre enn ett millisekund skiller gjennomsnittene og medianene til spørringene. Ettersom begge er eksekvert i vekselvis rekkefølge og under like forutsetninger, er det derfor rimelig å anta at kompleksiteten til de to spør-

ringene er tilnærmet like.

De ovennevnte løsningene presenterer én måte å kombinere tabeller på, men dette kan gjøres på flere måter. Listingene 4.8 og 4.9 viser hvordan spørringene kan konstrueres med bruk av *explicit joins*. Kommandoen *join* [62] produserer nye rader ved å sammenligne hver rad i den første tabellen med hver rad i den andre. Ved å kombinere tabellene på tidspunkt og den gitte avstanden mellom spillerne, samt behandle resten parametrene i en *where*-klausul, returneres det samme resultatet som de ovennevnte spørringene gjør. Kjøretidene presenteres i tabell 4.3.

Listing 4.8: Eksempel 4 - *Avstand mellom to spillere* i pseudokode

```
SELECT one.player, two.player, one.time-10 sec,
       one.time+10 sec
FROM Match one
JOIN Match two
one.pos IN circle(two.pos,distance) AND
one.time = two.time
WHERE  ...
```

Listing 4.9: Eksempel 5 - *Avstand mellom to spillere* i pseudokode

```
SELECT one.player, two.player, one.time-10 sec,
       one.time+10 sec
FROM <subquery> one
JOIN <subquery> two
one.pos IN circle(two.pos,distance) AND
one.time = two.time
```

Ideen med oppretting av skjemaer via kommandoen *with* eller som subspørringer, er å begrense antall rader som må itereres gjennom og kombineres i det ytre leddet av spørringen. Formålet er å senke kompleksiteten. Resultatene viser imidlertid at løsningen (listing 4.8) som bruker kamptabellen til å definere to skjemaer og deretter *join* for å kombinere dem, produserer kortest gjennomsnittlig kjøretid. Spørringen som inneholder subspørringer bruker derfor lengre tid på opprette skjemaene enn det som tjenes inn når kombinering skal utføres. Et bidragsytende element kan være at antall operasjoner som utføres totalt sett er relativt lite, og dette påvirkes av frontend-delen av systemet. Ettersom brukergrensesnittet ikke støtter markering av flere elementer for samme parameter, utføres ingen *OR*-operasjoner i det ytre leddet av spørringen. På bakgrunn av testresultatene er derfor løsningen i listing 4.8 implementert i analyseverktøyet.

Tabell 4.3: Sammenligning av spørringene i listing 4.8 og 4.9

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 4 (4.8)	269,222 ms, 279,920 ms 281,496 ms, 281,647 ms 284,364 ms, 285,380 ms 286,299 ms, 288,278 ms 292,415 ms, 294,342 ms	284,3363 ms	284,872 ms
Eksempel 5 (4.9)	281,684 ms, 284,563 ms 284,869 ms, 285,995 ms 286,747 ms, 288,042 ms 289,997 ms, 292,185 ms 292,752 ms, 296,481 ms	288,3315 ms	287,3945 ms

4.3.3 Sprinter

Bakgrunn

Helhjertede løp er nødvendig i både forsvarende og angripende situasjoner. Bevegelse skaper rom, og åpne rom er hensiktsmessige for å opprettholde flytende angrepsspill og produsere målsjanser. Kantspillere skal løpe opp og ned langs vingen av banen, og angrepsspillere skal skape rom for seg selv og medspillerne. For spillere med defensive plikter, er hurtighet også en fordel. Motspillerne skal følges og dekkes, og forsvarerne skal forhindre at motstanderne kommer til målsjanser.

I etterkant av kampen mellom Brann og Sandnes Ulf under 2014-sesongen i Tippeligaen, uttalte Branns kaptein Erlend Hanstveit at Brann-spillerne 'løp litt for lite og lot Sandnes Ulf få kontre' [10]. Løpskraft, hurtighet og innsats kan utgjøre forskjellen mellom seier og tap i en fotballkamp, og helhjertede løp kan ofte være en indikasjon på at kritiske situasjoner enten oppstår eller avverges. *All sprints faster than 10 mps* er den tredje og siste spørringen som presenteres i [41] og finner alle situasjoner hvor en spesifikk spiller oppnår en løpshastighet på 10 meter per sekund eller mer. Denne spørringen ignorerer imidlertid aspekter knyttet til varighet og retning av løpet. I offensive situasjoner er det viktig at angrepsspillere fullfører løpene sine. Selv om de ikke mottar ballen, vil løpene skape rom for medspillerne. Fullføring av løp er muligens enda viktigere når motstanderlaget forsøker å kontre. For å forhindre produsering av farlige situasjoner, må spillerne kjapt komme seg på rett side av ballen.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **direction** (float) - Retning spiller løper mot. Verdien 0 er y-aksens retning
- **velocity** (float) - Spillers løpshastighet i meter per sekund

Kombinert inneholder disse attributtene informasjon om spillernes løpsretning og -hastighet. Dette danner bakgrunnen for spørringen *Sprinter* som finner alle situasjoner hvor en spesifikk spiller løper i en bestemt retning og hastighet i et minimum antall sekunder.

Frontend

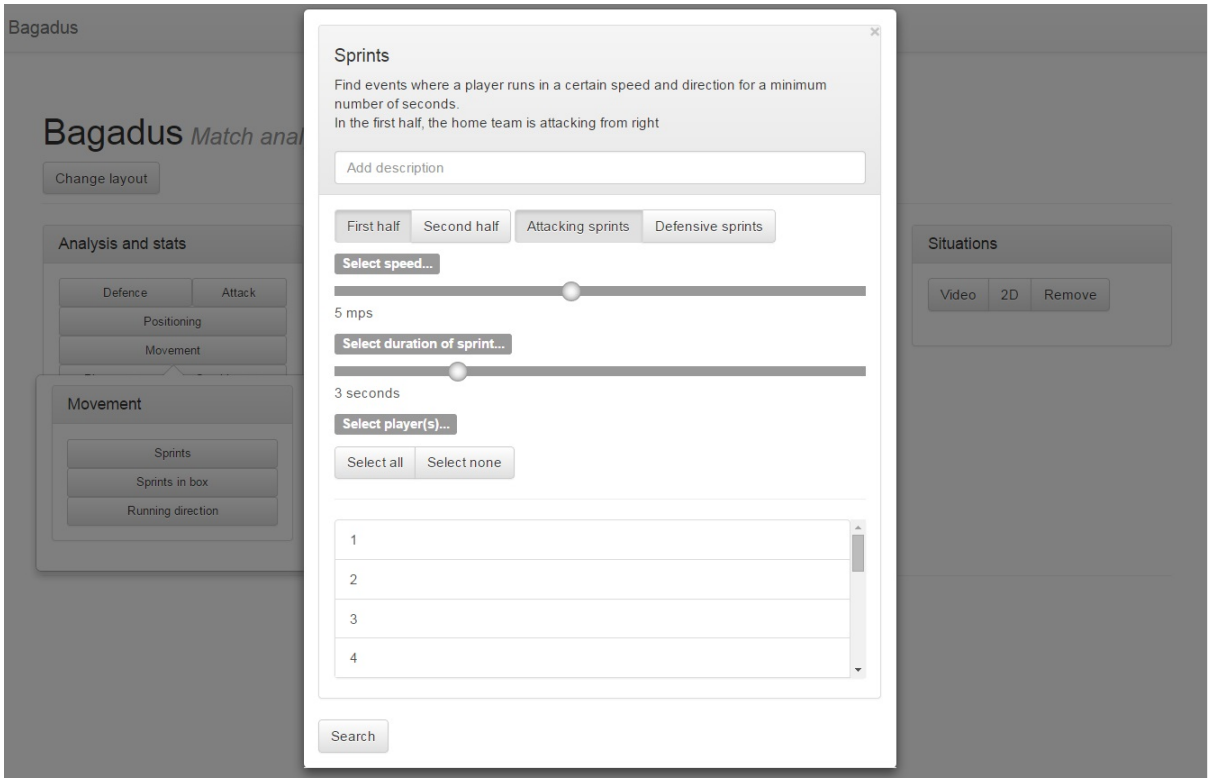
Spørringen *Sprinter* skal fungere som et virkemiddel i analysering av spilleres bevegelsesmønstre. I kampanalysesiden er den derfor representert i kategorien *Movement* (bevegelse). Popup-vinduet som vises i figur 4.3 inkluderer følgende elementer for definering av parametrene:

- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Løpsretning**: To knapper navngitt *Attacking* og *Defending*.
- **Løpshastighet**: En *slider* med rekkevidde fra 1 til 10 meter per sekund
- **Løpsvarighet**: En *slider* med rekkevidde fra 1 til 10 sekunder
- **Spiller(e)**: En liste med spilleridentifikasjoner som støtter markering av flere elementer

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Sprinter* til databaseserveren, tar åtte argumenter:

- Navnet på kamptabellen
- En tallverdi som definerer varigheten av løpet
- En tekststreng (i form av enten *attacking* eller *defending*) som definerer retningen av løpet
- En tallverdi som definerer løpshastigheten
- En liste med spillere
- En tekststreng som definerer hvilken omgang som skal analyseres
- En tekststreng som definerer hvilken banehalvdel som hjemmelaget angriper fra i første omgang



Figur 4.3: Popup for setting av parametre for spørringen *Sprinter*

- Kampens starttidspunkt

Projeksjonene i løsningene som presenteres under, består av tre kolonner som inkluderer spilleridentifikasjon og to tidspunkt. Tidspunktene brukes for å hente video, mens spilleridentifikasjonen inkluderes for å opplyse bruker om hvilken spiller hver situasjon er knyttet til.

For å finne alle situasjoner hvor en bestemt spiller løper i en bestemt retning og hastighet i et minimum antall sekunder, sammenlignes parametrene med følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **direction** (float) - Retning spiller løper mot. Verdien 0 er y-aksens retning
- **velocity** (float) - Spillers løpshastighet i meter per sekund

Attributtet *direction* spesifiserer hvilken retning hver spiller løper i. Dersom en spiller løper fra høyre til venstre på banen (fra kameraenes perspektiv), vil verdien av dette attributtet være mindre enn 0. Verdien vil være større enn 0 dersom en spiller løper i motsatt retning. Parametrene som angår hvilken side hjemmelaget angriper fra i første

Tabell 4.4: Verdier for *retning* basert på parametrene

Retning	Omgang	Side i første omgang	Retning
>0	første	venstre	angripende
	første	høyre	forsvarende
	andre	venstre	forsvarende
	andre	høyre	angripende
<0	første	venstre	forsvarende
	første	høyre	angripende
	andre	venstre	angripende
	andre	høyre	forsvarende

omgang, valg av omgang og hvorvidt løpet er av angripende eller forsvarende natur, brukes til å definere spørringsbetingelsen knyttet til løpsretningen (se tabell 4.4 og listing 4.10).

Listing 4.10: Java-kode som setter verdien til attributtet *direction*

```
if((half.equals(‘first‘) &&
  sideInFirstHalf.equals(‘left‘) &&
  sprintType.equals(‘attacking‘)) || ...) {
  directionComparator = ‘>0‘;

} else {
  directionComparator = ‘<0‘;
}
```

Informasjon om hvor fort hver spiller løper, lagres 20 ganger i sekundet i databasen. For å ta stilling til varigheten av et løp, må det derfor implementeres en betingelse som bestemmer at det må finnes 20 rader per sekund under hele løpsvarigheten hvor spilleren holder den gitte løpshastigheten. Dersom løpsvarigheten settes til fire sekunder og hastigheten til seks meter per sekund, må det derfor eksistere 20*4 rader hvor spilleren løper i seks meter per sekund innenfor et tidsintervall på fire sekunder.

Projeksjonen i spørringen som vises i listing 4.11, henter informasjon fra to subspørringer. Den første subspørringen finner alle situasjoner hvor en gitt spiller løper fortere enn en viss hastighet i en bestemt retning. Den andre subspørringen er identisk til den første med unntak av at den returnerer et tidsintervall i stedet for et tidspunkt. Dette tidsintervallet har en varighet som tilsvarer den brukerdefinerte løpsvarigheten.

Ved å gruppere returverdiene på spillerattributter fra tabell én og start- og sluttidspunkt fra tabell to, kan antall forekomster av de ønskede situasjonene telles ved hjelp av PostgreSQL sin innebygde funksjon *count()* [87]. Ettersom *where*-klausuler ikke kan benyttes ved funksjoner som samler verdier (aggregat-funksjoner), defineres betingelsen tilknyttet antall forekomster, i en *having*-klausul [88]. Tidspunktene som spørringen finner avrundes til nærmeste tiende sekund og brukes til å definere tidsintervaller som tilsvarer lengden av løpet addert med ti sekunder før og etter situasjonen.

Listing 4.11: Eksempel 1 - *Sprinter* i pseudokode

```
SELECT a.player, round_time(b.start,10)-10 sec,
       round_time(b.end,10)+10 sec, count(a.velocity)
FROM
    (SELECT player, time as start, velocity
     FROM Match
     WHERE velocity >= speed AND
     direction > dir AND
     player = player AND
     time > start_time + 60 min) a,
    (SELECT player, time as start, time+duration sec as end
     FROM Match
     WHERE velocity >= speed AND
     direction > dir AND
     player = player AND
     time > start_time + 60 min) b
 WHERE a.start BETWEEN b.start AND b.end AND
 a.player = b.player
 GROUP BY a.player, round_time(b.start,10)-10 sec,
          round_time(b.end,10)+10 sec
 HAVING count(a.velocity)>=20*duration
```

Løsningen i listing 4.12 viser et alternativ til den ovennevnte metoden for definering av skjemaer. Ved hjelp av kommandoen *with*, opprettes to midlertidige skjemaer, og disse kombineres deretter i en *where*-klausul. Kommandoen *with* har tidligere blitt presentert i seksjon 4.3.2 om spørringen *Avstand mellom to spillere*, og i det tilfellet medførte eksekvering lange kjøretider. Blant attributtene som ble returnert av skjemaene, var spillernes posisjoner, et attributt som ikke finnes i skjemaene som vises i listing 4.12. Mangel på indeksering av elementer ble i seksjon 4.3.2 fremlagt som en mulig årsak til de lange kjøretidene. På bakgrunn av forskjellen i projeksjonene, er det derfor plausibelt at løsningen

under fullfører med kortere kjøretider.

Listing 4.12: Eksempel 2 - *Sprinter* i pseudokode

```
WITH a AS
     (SELECT player, time as start, velocity
      FROM Match
      WHERE velocity >= speed AND
      direction > dir AND
      player = player AND
      time > start_time + 60 min),
b AS
     (SELECT player, time as start, time+duration sec as end
      FROM Match
      WHERE velocity >= speed AND
      direction > dir AND
      player = player AND
      time > start_time + 60 min)
SELECT a.player, round_time(b.start,10)-10 sec,
       round_time(b.end,10)+10 sec, count(a.velocity)
FROM a, b
WHERE a.start BETWEEN b.start AND b.end AND
      a.player = b.player
GROUP BY a.player, round_time(b.start,10)-10 sec,
      round_time(b.end,10)+10 sec
HAVING count(a.velocity)>=20*duration
```

Ideen med spørringene i listing 4.11 og 4.12 er å definere skjemaer som utelukkende returnerer informasjon innenfor parametrenes grenser. Motivasjonen er å begrense mengden informasjon som må behandles og sammenlignes i den ytre spørringen, og dermed senke kompleksiteten. Listing 4.13 viser et alternativ til denne fremgangsmåten og presenterer en løsning som potensielt kan lide av lengre kjøretider. To skjemaer er definert, hvor ett er kamptabellen og det andre er en subspørring. Betingelser tilknyttet kamptabellen defineres deretter i den ytre spørringen.

Listing 4.13: Eksempel 3 - *Sprinter* i pseudokode

```
SELECT a.player, round_time(b.start,10)-10 sec,
       round_time(b.end,10)+10 sec, count(a.velocity)
FROM Match a,
    (SELECT player, time as start, time+duration sec as end
```

```
      FROM Match
      WHERE velocity >= speed AND
      direction > dir AND
      player = player AND
      time > start_time + 60 min) b
WHERE a.time BETWEEN b.start AND b.end AND
      a.player = b.player AND a.velocity >=  speed AND
      a.direction  > dir
GROUP BY a.player, round_time(b.start,10)-10 sec,
      round_time(b.end,10)+10 sec
HAVING count(a.velocity)>=20*duration
```

Testing

For å sammenligne løsningene, har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze*. Parametrene er predefinerte og presenteres under:

- **Kamp**: Tromsø-Tottenham
- **Spiller**: 1
- **Løpshastighet**: 5 meter per sekund
- **Løpsvarighet**: 3 sekunder
- **Løpsretning**: Angripende
- **Omgang**: første

Testresultatene i tabell 4.5 viser at medianene og de gjennomsnittlige kjøretidene er marginalt ulike (opptil 3 millisekunder skiller dem). I dette tilfellet foreligger det derfor ingen signifikante forskjeller mellom de tre teknikkene for definering av skjemaer, et liknende resultat til det som presenteres om spørringen *Avstand mellom to spillere* i seksjon 4.3.2. En årsak til dette kan være at parameter-rekkeviddene er små, og dette bidrar til at færre operasjoner må utføres og mindre informasjon må kombineres i det ytre leddet av spørringen.

Popup-vinduet som håndterer definering av parametre i brukergrensesnittet, støtter valg av flere spillere, og spørringen vil da finne separate situasjoner for hver spiller. Dette øker mengden informasjon som returneres av de definerte skjemaene og som må behandles i det ytre leddet av spørringen. I tillegg vil kommandoen *OR* medføre et økt antall operasjoner som utføres, et aspekt som kan prege kjøretidene. For å oppnå et større testgrunnlag har

Tabell 4.5: Sammenligning av spørringene i listing 4.11, 4.12 og 4.13 med én spiller som parameter

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.11)	351,433 ms, 354,911 ms 355,982 ms, 356,049 ms 356,649 ms, 356,781 ms 357,083 ms, 361,993 ms 364,383 ms, 366,629 ms	358,1893 ms	356,715 ms
Eksempel 2 (4.12)	333,878 ms, 347,066 ms 351,137 ms, 352,208 ms 353,194 ms, 358,450 ms 358,821 ms, 360,137 ms 362,202 ms, 373,885 ms	355,0978 ms	355,822 ms
Eksempel 3 (4.13)	350,272 ms, 350,319 ms 353,166 ms, 355,058 ms 355,244 ms, 356,586 ms 356,734 ms, 359,183 ms 361,006 ms, 370,084 ms	356,7652 ms	355,915 ms

derfor et nytt sett med betingelser blitt definert. Dette settet er identisk til det første med unntak av parameteret tilknyttet spillere (se Vedlegg C for fullstendig kode):

- **Spiller**: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 eller 14

Når spørringene eksekveres med flere spillerparametre, synliggjøres større forskjeller mellom løsningene enn det som presenteres i tabell 4.5. Tabell 4.6 viser at løsningen som bruker kamptabellen til å definere et av skjemaene, er opptil 300 millisekunder tregere enn de to andre. I dette tilfellet kan det derfor konkluderes med at oppretting av skjemaer via kommandoen *with* eller som subspørringer, er hensiktsmessig i søken etter raske løsninger.

Løsningen som oppretter skjemaer via kommandoen *with* (se listing 4.12) er raskest, og dette står i sterk kontrast til testresultatene i seksjon 4.3.2 (*Avstand mellom to spillere*). Ettersom skjemaene lagres midlertidig i minnet, er det nødvendig at attributtene i projeksjonen er indeksert og kan slås opp raskt. Løsningen som vises i listing 4.12 inkluderer ikke attributtet *position* i projeksjonen. Det er derfor realistisk å anta at mangel på indeksering av dette attributtet, bidrar til de høye kjøretidene som presenteres i seksjon 4.3.2.

Tabell 4.6: Sammenligning av spørringene i listing 4.11, 4.12 og 4.13 med flere spillere i parameterlisten

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.11)	2088,254 ms, 2091,115 ms 2092,792 ms, 2094,870 ms 2096,535 ms, 2103,745 ms 2115,858 ms, 2120,097 ms 2126,260 ms, 2134,039 ms	2106,3565 ms	2100,14 ms
Eksempel 2 (4.12)	1997,214 ms, 1998,670 ms 1999,818 ms, 1999,928 ms 2001,297 ms, 2001,913 ms 2004,589 ms, 2012,264 ms 2020,298 ms, 2028,303 ms	2006,4294 ms	2001,605 ms
Eksempel 3 (4.13)	2279,964 ms, 2280,346 ms 2280,412 ms, 2285,760 ms 2287,817 ms, 2292,773 ms 2293,716 ms, 2298,994 ms 2305,927 ms, 2341,904 ms	2294,7613 ms	2290,295 ms

Løsningene som har blitt presentert så langt, definerer skjemaer på ulike måter men bruker samme metode for kombinering (via *where*-klausuler). Eksemplene under presenterer et alternativ til denne kombineringsteknikken og er reviderte versjoner av løsningene som vises listing 4.11 og 4.12. Via kommandoen *join* kombineres de definerte skjemaene på spilleridentifikasjon og tid.

Listing 4.14: Eksempel 4 - *Sprinter* i pseudokode

```
SELECT a.player, round_time(b.start,10)-10 sec,
       round_time(b.end,10) +10 sec, count(a.velocity)
FROM <subquery> a
JOIN <subquery> b
ON a.start BETWEEN b.start AND b.end AND
a.player = b.player
GROUP BY  a.player, round_time(b.start,10)-10 sec,
          round_time(b.end,10) +10 sec
HAVING count(a.velocity)>(20* duration)
```

Listing 4.15: Eksempel 5 - *Sprinter* i pseudokode

```
WITH a AS <subquery one>,
     b AS <subquery two>
```

```
SELECT a.player, round_time(b.start,10)-10 sec,
       round_time(b.end,10) +10 sec, count(a.velocity)
FROM a
JOIN b ON
     a.start BETWEEN b.start AND b.end AND
     a.player = b.player
GROUP BY  a.player, round_time(b.start,10)-10 sec,
     round_time(b.end,10) +10 sec
HAVING count(a.velocity)>(20* duration)
```

Testresultatene (se tabell 4.6 og 4.7) viser marginale forskjeller mellom kombineringsteknikkene *where* og *join*. Dette resultatet vises også i seksjon 4.3.2. Eventuelle forskjeller mellom de to teknikkene, virker derfor å utelukkende være tilknyttet lesbarhet av kode, en teori som fremlegges i fora som [30] og [29]. Videre er det opptil 100 millisekunder i forskjell mellom de to teknikkene for oppretting av skjemaer (*with* vs subspørring). Løsningen som benytter kommandoene *with* og *where* for henholdsvis oppretting og kombinering av skjemaer, har kortest gjennomsnittlig kjøretid. På bakgrunn av dette er derfor spørringen i listing 4.12 implementert i analyseverktøyet.

Tabell 4.7: Sammenligning av spørringene i listing 4.14 og 4.15 med flere spillere i parameterlisten

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 4 (4.14)	2071,805 ms, 2087,382 ms 2087,853 ms, 2087,946 ms 2093,662 ms, 2094,624 ms 2095,587 ms, 2103,760 ms 2116,009 ms, 2121,545 ms	2096,0173 ms	2094,143 ms
Eksempel 5 (4.15)	1991,025 ms, 1991,813 ms 1998,602 ms, 2004,676 ms 2006,341 ms, 2007,125 ms 2013,590 ms, 2014,186 ms 2034,069 ms, 2035,218 ms	2009,6645 ms	2006,733 ms

4.3.4 Sprinter i område

Bakgrunn

Spillere i ulike posisjoner har ulike plikter, defensivt og offensivt. Angripende strategier som for eksempel involverer innlegg fra kantene, krever at både kantspillerne og spissene er i stand til å posisjonere seg riktig. Et implisitt aspekt i produsering av målsjanser av den typen, er at spillerne kan frigjøre seg fra motstanderne og skape rom for seg selv. Dette forutsetter at spillerne beveger seg hurtig i de avgjørende sonene på banen. Hurtighet i avgjørende soner og øyeblikk er også viktig i forsvarende situasjoner. Når en potensiell målsjanse oppstår, må forsvarerne bevege seg hurtig inn i faresonen og avverge situasjonen.

De tidligere forklarte spørringene *Spiller i område* og *Sprinter* finner to forskjellige typer kampsituasjoner, henholdsvis i kategoriene *Positioning* (posisjonering) og *Movement* (bevegelse). *Sprinter i område* er en utvidelse av de to konseptene og finner alle situasjoner hvor en spesifikk spiller oppnår en bestemt løpshastighet innenfor et avgrenset område på banen.

Frontend

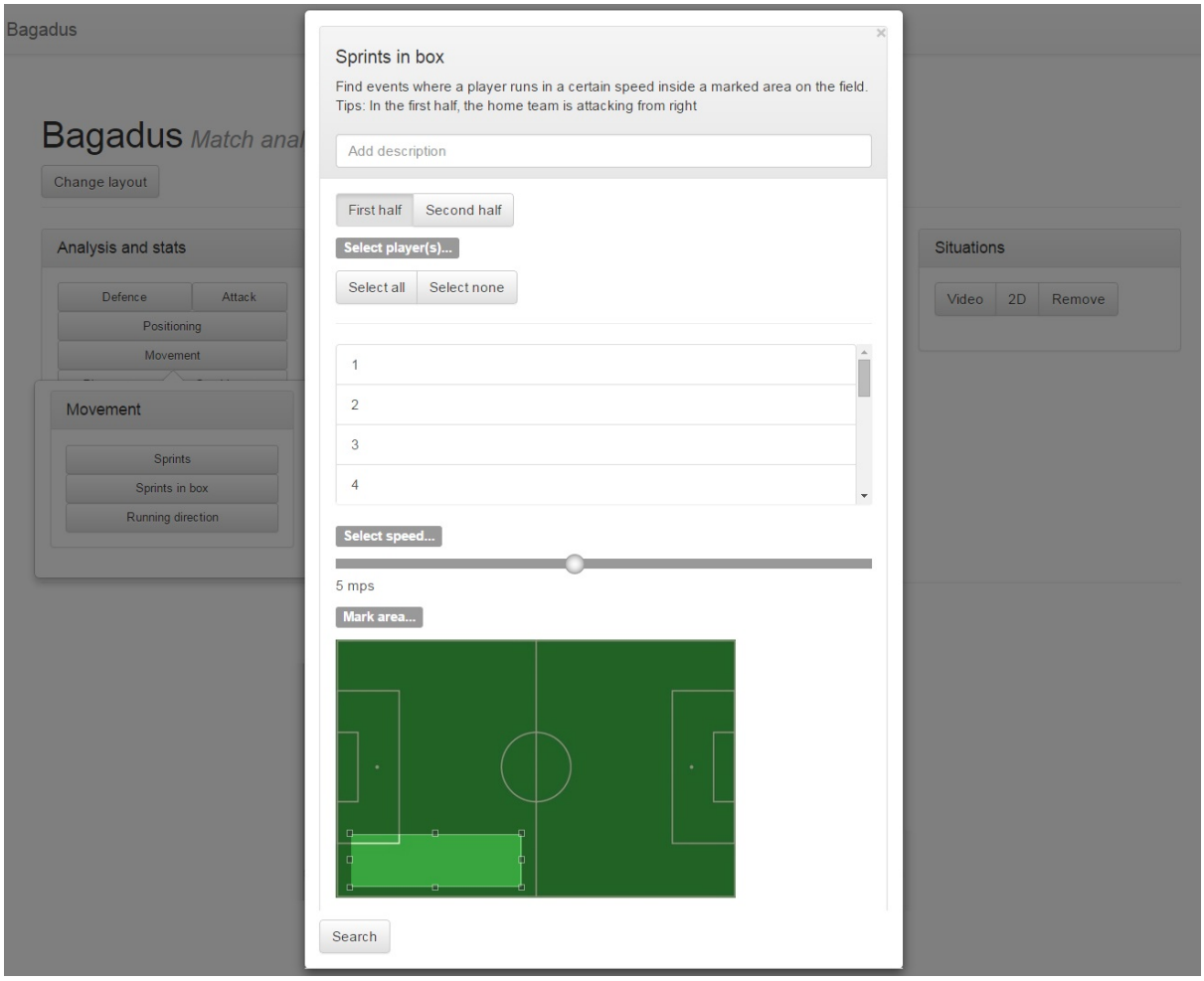
Spørringen *Sprinter i område* skal fungere som et virkemiddel i analysering av hvordan spillere beveger seg i bestemte soner på banen. I kampanalysesiden er den derfor representert i kategorien *Movement* (bevegelse). Popup-vinduet som vises i figur 4.4 inkluderer følgende elementer for definering av parametre:

- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Løpshastighet**: En *slider* med rekkevidde fra 1 til 10 meter per sekund
- **Spiller(e)**: En liste med spilleridentifikasjoner som støtter markering av flere elementer
- **Område**: Markering av felt på en opptegnet fotballbane

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Sprinter i område* til databaseserveren, tar åtte argumenter:

- Navnet på kamptabellen
- En liste med spillere



Figur 4.4: Popup for setting av parametre for spørringen *Sprinter i område*

- Fire koordinater som definerer det avgrensede området
- Løpshastigheten
- En tekststreng som definerer hvilken omgang som skal analyseres
- Kampens starttidspunkt

Projeksjonen i løsningen som presenteres under, inneholder tre kolonner som inkluderer spilleridentifikasjon og to tidspunkt. Tidspunktene, som sammen danner et tidsintervall, brukes for å hente video, mens spilleridentifikasjonen inkluderes for å opplyse bruker om hvilken spiller hver situasjon er knyttet til.

For å finne alle situasjoner hvor en spesifikk spiller oppnår en bestemt løpshastighet i et avgrenset område på banen, sammenlignes parametrene med følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid

- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen
- **velocity** (float) - Spillers løpshastighet i meter per sekund

Løpsforsøk som har et angripende formål behøver ikke å ha en bestemt retning. Ved innlegg fra kantene, vil en angrepsspiller bevege seg hurtig dit ballen havner. I forsvarende situasjoner vil de defensive spillerne forsøke å følge motspillerne, og løpsretningene kan dermed peke i alle retninger. På bakgrunn av dette, inkluderes ikke attributtet *direction* (retning) i spørringen.

Den konstruerte spørringen henter informasjon fra ett enkelt skjema, kamptabellen, og parametrene behandles i en *where*-klausul. Formålet med spørringen er å muliggjøre analysering av spilleres løpsforsøk i bestemte soner på banen, og den er derfor strukturmessig og til dels konseptuelt lik spørringen *Spiller i område*. Tidsintervallene som defineres har en lengde på 20 sekunder (10 sekunder før og etter), og i likhet med tidligere forklarte spørringer vil overlappende tidsintervaller slås sammen i kontrollermetoden. Dette sørger for at resultatene vil returnere hele situasjoner, uavhengig av varighet. Listing 4.16 viser spørringen i pseudokode, og parametrene er markert rødt i koden.

Listing 4.16: *Sprinter i område* i pseudokode

```
SELECT player, round_time(time,10)-10 sec, round_time(time,10)+10 sec
FROM  Match
WHERE (player =  player1 OR player =  player2 OR
       player =  player3) AND
       pos_x between  X1 AND  X2 AND
       pos_y between  Y1 AND  Y2 AND
       velocity > speed AND
       time > ( start_time + 60 minutes);
```

4.3.5 Løpebane

Bakgrunn

I prosessen om å komme til målsjanser, er evne til å frigjøre seg fra motspillere viktig, og attributter som akselerasjon, ekslosivitet og hurtighet spiller viktige roller. Det finnes imidlertid flere eksempler på offensive spillere med disse egenskapene som ikke produserer målsjanser, og dette kan være være grunnet mangel på pasningsleggere og kreative drivkrafter innad i laget. Et annet bidragsytende element kan være angrepsspillernes

manglende evner til å bevege seg uten ball. I offensive situasjoner er det essensielt at de angripende spillerne er i stand til å ane potensielle målsjanser. De må kunne tolke situasjoner og bruke denne informasjonen til å bevege seg i rom hvor kritiske situasjoner kan oppstå.

For å komme til målsjanser, er det hensiktsmessig at angrepsspillerne posisjonerer seg i soner som kan bidra til at ballinnehaver har pasningsalternativ. I figur 4.5 vises to bilder fra kampen mellom Barcelona og Real Madrid 21. april 2012. Situasjonen (som ledet til at Real Madrid skåret) viser at Christiano Ronaldo beveger seg inn i rommet bak Barcelonas forsvarsrekke og mottar ballen i en én-mot-én situasjon mot keeper. Hendelsen oppstår ved at Ronaldo befinner seg i en initial posisjon, og idet en mulighet øynes signaliserer han til ballinnehaver hvor han kommer til å løpe og hvor han vil ha ballen.

Figur 4.5: Skjermskudd av Christiano Ronaldos løp i kampen mellom Barcelona og Real Madrid 21. april 2012 [94]

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp**(String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id**(int) - Spilleridentifikator
- **position**(Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen 20 ganger i sekundet. Dette danner bakgrunnen for spørringen *Løpebane* som finner alle situasjoner hvor en spiller beveger seg fra en sone på banen til en annen på et bestemt antall sekunder.

Frontend

Spørringen *Løpebane* skal fungere som et virkemiddel i analysering av spilleres bevegelsemønstre. I kampanalysesiden er den derfor representert i kategorien *Movement* (bevegel-

se). Popup-vinduet som vises i figur 4.6 inkluderer følgende elementer for definering av parametre:

- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Spiller(e)**: En liste med spilleridentifikasjoner som støtter markering av flere elementer
- **Løpebane**: Tegning av pil fra en sone til en annen på en opptegnet fotballbane
- **Tidsbegrensning**: En *slider* med rekkevidde fra 1 til 30 sekunder

Den opptegnede fotballbanen er definert som et *canvas* [91] i HTML-koden, og koordinatene som oppnås når en pil trekkes fra et område til et annet på banen, lagres fortløpende. De tre Javascript-funksjonene som presenteres i listing 4.17, fungerer som lyttere til musepekeren. *Event*-objektet [86] som metodene tar som argument, inneholder informasjon om musepekerens posisjon i *canvas*-objektet. Denne informasjon konverteres til koordinater på den ekte fotballbanen og lagres i to nye objekter. Avstanden mellom de to punktene kalkuleres i *onmouseup*-funksjonen som indikerer at musepekerens interaksjon med *canvas*-objektet er ferdig.

Listing 4.17: Javascript-funksjoner for registrering av musepekerens interaksjon med et *canvas*-objekt

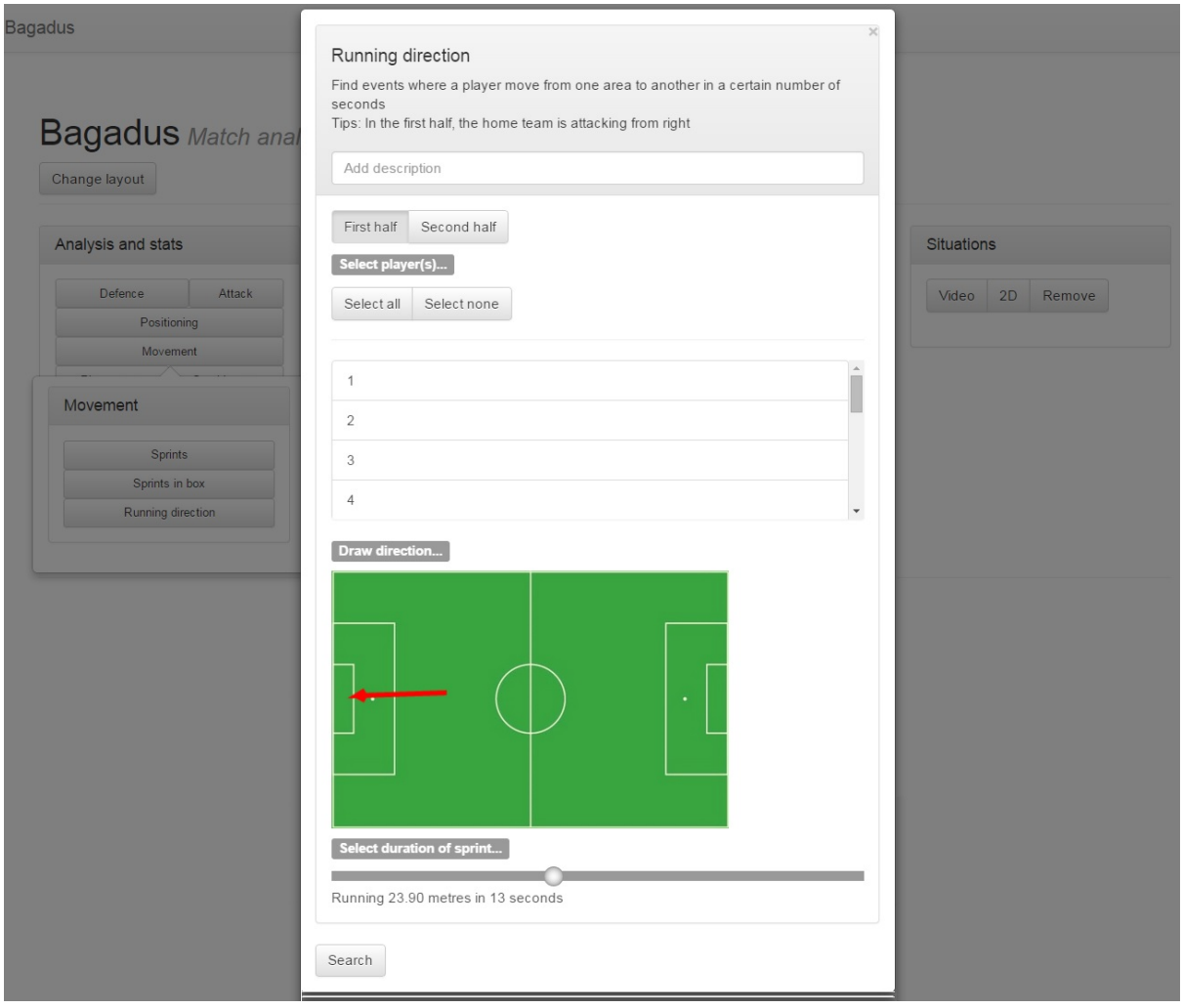
```
var el = document.getElementById('canvas');

el.onmousedown = function(e) {
  $scope.start = {'x': e.pageX*(105/$scope.canvasWidth),
                  'y': e.pageY*(68/$scope.canvasHeight)};
};
el.onmousemove = function(e) {
  $scope.stop = {'x': e.pageX*(105/$scope.canvasWidth),
                 'y': e.pageY*(68/$scope.canvasHeight)};
};
el.onmouseup = function(e) {
  $scope.distance =
  Math.sqrt(($scope.stop.x - $scope.start.x)^2 +
  ($scope.stop.y -$scope.start.y)^2);
};
```

Før parametrene sendes til backend-delen av systemet, defineres to kvadratiske områder med start- og sluttpunktene som sentrum i hvert sitt område (se listing 4.18). Spørringene under vil derfor inneholde klausuler som stadfester at den gitte spilleren må bevege seg

fra et sted i startområdet til et sted i sluttområdet innenfor den gitte tidsbegrensningen, og parametrene vil inkludere totalt åtte koordinater som avgrenser to områder på banen.

Listing 4.18: Definering av startområde

```
var X1 = $scope.start.x -5;
var X2 = $scope.start.x +5;
var Y1 = $scope.start.y -5;
var Y2=  $scope.start.y +5;
```



Figur 4.6: Popup for setting av parametre for spørringen *Løpebane*

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Løpebane* til databaseserveren, tar 13 argumenter:

- Navnet på kamp-tabellen
- En liste med spillere
- Åtte koordinater som definerer to avgrensede områder
- Den maksimale tiden en spiller kan bruke på å forflytte seg fra et område til et annet
- En tekststreng som definerer hvilken omgang som skal analyseres
- Kampens starttidspunkt

Projeksjonene i løsningene som presenteres under består av tre kolonner med spilleridentifikasjon og to tidspunkt. De to tidspunktene brukes for å hente video, mens spilleridentifikasjonen inkluderes for å opplyse bruker om hvilken spiller hver situasjon er knyttet til.

For å finne alle situasjoner hvor en bestemt spiller beveger seg fra et område til et annet på banen innenfor en begrenset periode av tid, sammenlignes parametrene med følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Den konseptuelle idéen bak spørringen *Løpebane*, er basert på to situasjoner og den tidsmessige avstanden mellom dem. Den første situasjonen representerer den gitte spillerens initiale posisjon på banen, mens den andre representerer sluttposisjonen. I listing 4.19 defineres hver situasjon som en subspørring. Ved hjelp av en *where*-klausul kombineres de to skjemaene på spilleridentifikasjon og den maksimale tiden mellom situasjonene, og hver subspørring tilsvarer spørringen *Spiller i område*.

Subspørringene returnerer hver sine tidspunkt, og sammen definerer de utgangspunktet for konstruksjon av tidsintervaller. Tidspunktene avrundes til nærmeste tiende sekund, og start- og sluttidspunktet henholdsvis subtraheres og adderes med 10 sekunder. På denne måten vil situasjonene inneholde spillernes løp fra start til slutt. I tillegg til overlappende resultater slås sammen i den aktuelle metoden i Java-kontrolleren.

Listing 4.19: Eksempel 1 - *Løpebane* i pseudokode

```
SELECT start.player, round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+10 sec
FROM (
      SELECT player, time
```

```
        FROM Match
        WHERE pos_x BETWEEN startX1 AND startX2 AND
        pos_y BETWEEN startY1 AND startY2 AND
        time > start_time + 60 min AND
        player = player) start,
       (SELECT player, time
        FROM Match
        WHERE pos_x BETWEEN stopX1 AND stopX2 AND
        pos_y BETWEEN stopY1 AND stopY2 AND
        time > start_time + 60 min AND
        player = player) stop
  WHERE stop.time > start.time AND
  stop.time < (start.time + time_limit sec) AND
  start.player = stop.player;
```

I likhet med tidligere forklarte spørringer, har ulike teknikker for definering av skjemaer blitt implementert. Oppretting av midlertidige skjemaer via kommandoen *with* har vist seg å produsere varierende resultater, avhengig av om attributtet *position* inkluderes i projeksjonen eller ikke. I subspørringene som vises i løsningen over, sammenlignes dette attributtet med parametre, men det utgjør ikke en del av radene som returneres. Det er derfor realistisk å anta kommandoen *with* i konteksten av spørringen *Løpebane*, kan produsere resultater med korte kjøretider. Listing 4.20 bruker denne kommandoen til å opprette skjemaer som er identiske til subspørringene i listing 4.19 og disse kombineres på spilleridentifikasjon og tid i en *where*-klausul.

Listing 4.20: Eksempel 2 - *Løpebane* i pseudokode

```
WITH start AS <subquery one>,
     stop AS <subquery two>
SELECT start.player, round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+10 sec
WHERE stop.time > start.time AND
      stop.time < (start.time + time_limit sec) AND
      start.player = stop.player;
```

Ideen bak skjemaene som defineres i de ovennevnte løsningene, er å utelukkende returnere informasjon som den ytre spørringen har behov for. Listing 4.21 viser en alternativ, og potensielt mer tidskostbar fremgangsmåte. Spørringen bruker kamptabellen til å definere to skjemaer, og samtlige betingelser behandles samme sted. Denne løsningen kan medføre et økt antall iterasjoner dersom samtlige rader i hver tabell sammenlignes med hverandre.

Dette er imidlertid avhengig av hvordan PostgreSQL optimaliserer spørringen.

Listing 4.21: Eksempel 3 - *Løpebane* i pseudokode

```
SELECT start.player, round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+10 sec
FROM Match start, Match stop
WHERE start.pos_x BETWEEN startX1 AND startX2 AND
      start.pos_y BETWEEN startY1 AND startY2 AND
      start.time > start_time + 60 min AND
      stop.pos_x BETWEEN stopX1 AND stopX2 AND
      stop.pos_y BETWEEN stopY1 AND stopY2 AND
      stop.time > start.time AND
      stop.time < (start.time + time_limit) AND
      start.player = player AND
      start.player = stop.player;
```

Testing

For å sammenligne løsningene (se Vedlegg D for fullstendig kode), har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze*. Parametrene er predefinerte og presenteres under:

- **Kamp**: Tromsø-Tottenham
- **Spiller**: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 eller 14
- **Startposisjon**: 27.15625 <X <37.15625, 35.7 <Y <45.7
- **Sluttposisjon**: 3.53125 <X <13.53125, 29.6625 <Y <39.6625
- **Maksimal forflytningstid**: 14
- **Omgang**: første

Løsningen som benytter kamptabellen til å definere to skjemaer, bruker omtrent tre ganger så lang tid på å fullføre som de to andre løsningene (se tabell 4.8). Dette står i stil med testresultatene som presenteres i seksjon 4.3.3 og viser at begrensning av informasjon som returneres av skjemaene, bidrar til lavere kompleksitet når de skal kombineres.

For å finne den mest optimale løsningen, må også ulike kombineringsteknikker testes. Løsningene under er basert på spørringene i listing 4.19 og 4.20 men kombinerer i stedet skjemaer via kommandoen *join*. Resultatene vises i tabell 4.9.

Tabell 4.8: Sammenligning av spørringene i listing 4.19, 4.20 og 4.21

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.19)	964,906 ms, 966,332 ms 966,494 ms, 972,178 ms 975,035 ms, 981,934 ms 985,896 ms, 990,541 ms 998,958 ms, 1007,132 ms	980,9406 ms	978,4845 ms
Eksempel 2 (4.20)	957,996 ms, 958,637 ms 960,607 ms, 965,874 ms 971,711 ms, 973,044 ms 976,575 ms, 978,026 ms 978,410 ms, 980,890 ms	970,177 ms	972,3775 ms
Eksempel 3 (4.21)	2679,404 ms, 2706,238 ms 2708,761 ms, 2712,316 ms 2726,252 ms, 2735,597 ms 2735,836 ms, 2737,304 ms 2750,910 ms, 2769,729 ms	2726,2347 ms	2730,9245 ms

Listing 4.22: Eksempel 4 - *Løpebane* i pseudokode

```
SELECT start.player, round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+10 sec
FROM <subquery one> start
JOIN <subquery two> stop
ON stop.time > start.time AND
   stop.time < (start.time + time_limit sec) AND
   start.player = stop.player;
```

Listing 4.23: Eksempel 5 - *Løpebane* i pseudokode

```
WITH start AS <subquery one>
     stop AS <subquery two>
SELECT start.player, round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+10 sec
FROM start
JOIN stop
ON stop.time > start.time AND
   stop.time < (start.time + time_limit sec) AND
   start.player = stop.player;
```

Testresultatene (se tabell 4.8 og 4.9) viser at de to løsningene (listing 4.20 og 4.23) som benytter *with* for å opprette skjemaer, produserer rader med kortest kjøretider. Videre har bruk av kombineringsteknikken *join* medført en gjennomsnittlig kjøretid som er fem millisekunder kortere enn spørringen som har kombinert via kommandoen *where*. Løsningen i listing 4.23 er derfor implementert i analyseverktøyet.

Tabell 4.9: Sammenligning av spørringene i listing 4.22 og 4.23

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 4 (4.22)	961,690 ms, 962,285 ms 965,665 ms, 974,416 ms 975,258 ms, 975,200 ms 979,061 ms, 979,145 ms 979,896 ms, 983,338 ms	973,5954 ms	975,229 ms
Eksempel 5 (4.23)	945,773 ms, 952,729 ms 953,792 ms, 958,995 ms 962,657 ms, 967,642 ms 975,334 ms, 976,132 ms 976,898 ms, 998,792 ms	966,8744 ms	965,1495 ms

4.3.6 Gruppering av spillere

Bakgrunn

Det finnes flere ulike situasjoner hvor spillere blir tvunget til eller velger å posisjonere seg i bolker på banen. I kamper hvor motstander er et betydelig bedre fotballag, er taktikker basert på kontringer ofte en foretrukket tilnærming. Dette innebærer at spillerne må ligge dypere i banen og sørge for at motstanderne har færre og mindre rom å spille seg gjennom. I offensive strategier er det naturlig at spillerne søker fremover og presser motstanderlaget høyt. Dette vil medføre at et flertall spillere grupperer seg høyere i banen. I tillegg kan gruppering av spillere kjennetegne dødballsituasjoner i form av frispark eller hjørnespark. Ved slike situasjoner vil et bestemt antall spillere befinne seg innenfor en 16-metersboks.

Den tidligere forklarte spørringen *Spiller i område* baserer seg på hvordan individuelle spillere posisjonerer seg på banen. Spørringen støtter valg av flere spillere, men den vil kun finne situasjoner hvor en av dem befinner seg innenfor det brukerdefinerte området. Spørringen som presenteres i denne seksjonen baserer seg på liknende parametre, men presenterer et middel for analyse av situasjoner hvor kollektiv posisjonering og gruppe-

ring av spillere er hovedfokuset.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen. Dette danner bakgrunnen for en spørring som finner alle situasjoner hvor et minimum antall spillere befinner seg innenfor et avgrenset område på banen.

Frontend

Gruppering av spillere skal fungere som et virkemiddel i analysering av hvordan spillere som et kollektiv posisjonerer seg på banen. I kampanalysesiden er den derfor representert i kategorien *Positioning* (posisjonering). Popup-vinduet som vises i figur 4.7 inkluderer følgende elementer for definering av parametre:

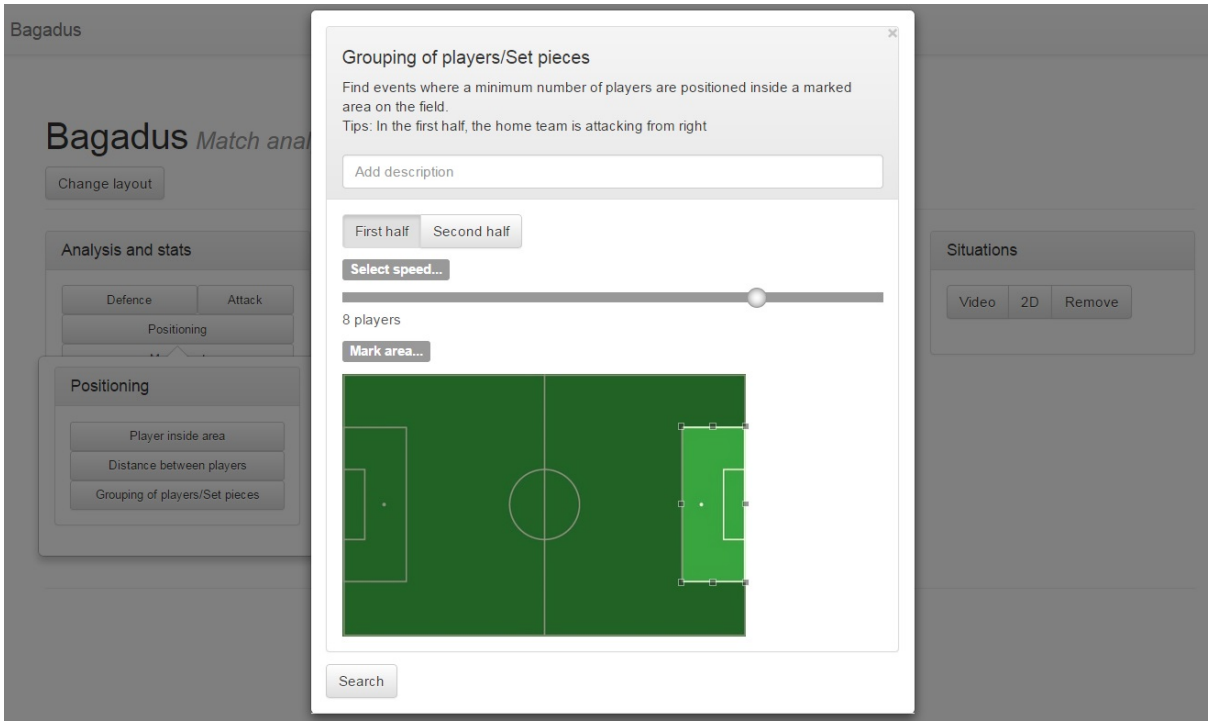
- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Antall spillere**: En *slider* med rekkevidde fra 1 til 10 spillere
- **Område**: Markering av felt på en opptegnet fotballbane

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Gruppering av spillere* til databaseserveren, tar åtte argumenter:

- Navnet på kamp-tabellen
- En tallverdi som bestemmer antallet spillere som må befinne seg i det avgrensede området
- Fire koordinater som definerer et avgrenset område på banen
- En tekststreng som definerer hvilken omgang som skal analyseres
- Kampens starttidspunkt

Ettersom betingelsene for spørringen er basert på spillergrupper av bestemte størrelser og ikke spesifikke enkeltspillere, inneholder ikke projeksjonen i løsningen under kolonner med spillerinformasjon. I stedet returnerer spørringen kun start- og sluttidspunkt som



Figur 4.7: Popup for setting av parametre for spørringen *Gruppering av spillere*

definerer et tidsintervall.

For å finne alle situasjoner hvor et minimum antall spillere befinner seg i et avgrenset område på banen, sammenlignes de brukerdefinerte parametrene med følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Spørringen som vises i listing 4.24, henter data fra ett skjema, kamptabellen, og en *where*-klausul spesifiserer hvilken omgang som skal analyseres og området som situasjonen skal oppstå i. Siste kolonnen i projeksjonen er et kall på PostgreSQL sin innebygde funksjon *count*, og parameteret er spilleridentifikasjonen i raden som behandles. Til slutt spesifiseres antallet spillere som må befinne i det definerte området, i en *having*-klausul.

Listing 4.24: *Gruppering av spillere - Dødballsituasjoner* i pseudokode

```
SELECT (round_time(time,10)-10 sec) as start,
       (round_time(time,10) + 10 sec) as stop, count(player)
FROM Match
WHERE pos_x BETWEEN  X1 AND X2 AND
```

```
pos_y BETWEEN Y1 AND Y2 AND
time > start_time+60 sec
GROUP BY round_time(time,10)
HAVING count(player) > nrOfPlayers;
```

4.3.7 Kollektivt forsvar og angrep

Bakgrunn

I forsvarende situasjoner er målet å ligge kompakt med laget og presse motstanderne tilstrekkelig til at ballen kan gjenvinnes. Gjenvinning av ball skaper mulighet for overganger, og kamptaktikken bidrar til å avgjøre hvor hurtig spillerne skal forflytte seg fremover i slike situasjoner. Kollektiv forflytning av spillere er også et viktig element når angrep resulterer i balltap. Når et lag mister ballen i en angripende situasjon, må spillerne med defensive plikter forflytte seg hurtig bakover på banen. Dette for å forhindre at motstanderne angriper i overtall mot et forsvar i ubalanse.

Ulike taktikker og strategier har ulike tilnærminger tilknyttet forflytning av spillere. Den kontringsbaserte taktikken involverer å ha mange spillere bak ballen i forsvarende situasjoner, og når muligheten byr seg, gjennomføre angrep hvor ballen og de offensive spillerne forflytter seg hurtig fremover på banen. Den ballbesittende strategien er i større grad basert på tålmodighet og går ut på å holde ballen i laget til eventuelle muligheter byr seg.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen. Dette danner bakgrunnen for et konsept som baserer seg på en initial sone og en sluttsone på banen. Et visst antall spillere skal på et tidspunkt befinne i startsonen, og på et senere tidspunkt skal et spesifikt antall spillere bevege seg inn i sluttsonen. Tilknyttet dette konseptet er det implementert to spørringer:

- En spørring som baserer seg på defensivt rettet spillerforflytning
- En spørring som baserer seg på offensivt rettet spillerforflytning

Frontend

I kampanalysesiden er spørringene *Kollektivt forsvar* og *Kollektivt angrep* henholdsvis representert i kategoriene *Defence* (forsvar) og *Attack* (angrep). Begge spørringene er basert på like argumenter, og popup-vinduene (se figur 4.8) for setting av parametre inkluderer følgende elementer:

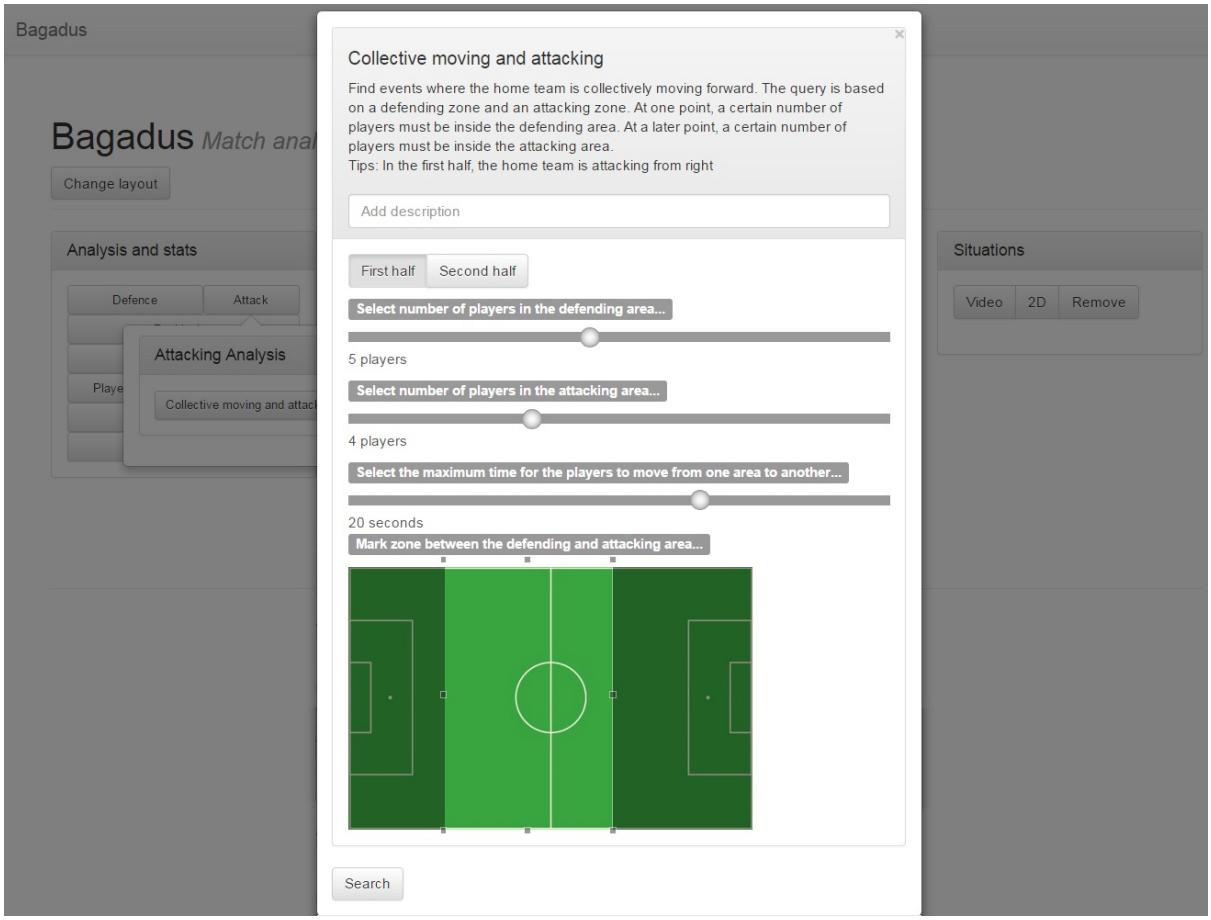
- **Omgang**: To knapper navngitt *First half* og *Second half*
- **Antall spillere i den defensive sonen**: En *slider* med rekkevidde fra 1 til 10 spillere
- **Antall spillere i den offensive sonen**: En *slider* med rekkevidde fra 1 til 10 spillere
- **Tid**: En *slider* med rekkevidde fra 1 til 30 sekunder
- **Område**: Markering av felt på en opptegnet fotballbane. Det markerte området definerer sonen mellom det defensive og det offensive feltet. Y-koordinatene er konstante og tilsvarer banens lengde på y-aksen. De to tilhørende x-koordinatene inkluderes i parameterlisten som sendes til Java-kontrolleren

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringene *Kollektivt forsvar* og *Kollektivt angrep* til databaseserveren, tar åtte argumenter:

- Navnet på kamptabellen
- En tallverdi som definerer antallet spillere som må befinne seg i det forsvarende området
- En tallverdi som definerer antallet spillere som må befinne seg i det angripende området
- Den maksimale tiden mellom hver situasjon
- To koordinater (på x-aksen) som definerer området mellom den forsvarende og den angripende sonen.
- En tekststreng som definerer hvilken omgang som skal analyseres
- En tekststreng som definerer hvilken side hjemmelaget angriper fra i første omgang

Projeksjonene i løsningene under, inneholder to kolonner med tidspunkt som kombinert definerer tidsintervaller. For å finne alle situasjoner hvor laget kollektivt forflytter seg bakover eller fremover på banen, sammenlignes parametrene med følgende attributter fra kamptabellen:



Figur 4.8: Popup for setting av parametre for spørringen *Kollektivt angrep*

- **timestamp**(String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id**(int) - Spilleridentifikator
- **position**(Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Konseptet er basert på to situasjoner og den tidsmessige avstanden mellom dem. De to situasjonene representeres av to områder på banen, et defensivt og et offensivt. Disse områdene defineres ved å markere én sone på banen. Sonen har predfinerte y-koordinater som dekker hele banens lengde på y-aksen, mens sonens lengde på x-aksen er brukerdefinert.

Det markerte feltet (se figur 4.8) utgjør området mellom den defensive og den offensive sonen. Hvilke koordinater som er tilknyttet hver av de to sonene, er avhengig av omgangen som skal analyseres og hvilken banehalvdel laget angriper fra i den gitte omgangen. Tabell 4.10 viser logikken, og denne logikken er uavhengig av hvilken spørring som eksekveres (*Kollektivt forsvar* eller *Kollektivt angrep*). Koordinatet *X1* er venstre kant av den markerte sonen (fra kameraenes perspektiv), mens *X2* er høyre kant.

Tabell 4.10: Definering av offensivt og defensivt område

Posisjon	Omgang	Side i første omgang
Offensivt område $<X1$ Defensivt område $>X2$	Andre Første	Venstre Høyre
Offensivt område $>X2$ Defensivt område $<X1$	Andre Første	Høyre Venstre

Løsningene som vises under, oppretter midlertidige skjemaer som utelukkende returnerer informasjon som er innenfor parametergrensene til hver situasjon. idéen bak dette har blitt presentert tidligere, og målet er å begrense antall rader som sammenlignes og kombineres i det ytre leddet av spørringen. Konseptet om kollektiv forflytning av spillere, kan implementeres uten bruk av skreddersydde skjemaer, men en slik fremgangsmåte har i tidligere seksjoner vist seg å medføre lange kjøretider. På bakgrunn av dette vil samtlige løsninger som presenteres i denne seksjonen, opprette midlertidige tabeller fremfor å definere hele kamptabellen som skjema.

Spørringen som vises i listing 4.25 finner situasjoner hvor spillerne forflytter seg i angripende retning i første omgang og hvor laget angriper fra venstre (se tabell 4.10). Hver situasjon er representert av en subspørring, og disse kombineres i en *where*-klausul som stadfester den maksimale tiden mellom hver situasjon. De brukerdefinerte parametrene er markert rødt i koden.

Listing 4.25: Eksempel 1 - *Kollektivt angrep* i pseudokode

```
SELECT round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+ 15 sec
FROM
      (SELECT time, count(player)
       FROM Match
       WHERE pos_x < X1 AND
       pos_y BETWEEN 0 AND 68 AND
       time < (start_time+55 min)
       GROUP BY time
       HAVING count(player) >= nrOfDefenders) start,
      (SELECT time, count(player)
       FROM Match
       WHERE pos_x > X2 AND
```

```
        pos_y BETWEEN 0 AND 68 AND
        time < (start_time+55 min)
        GROUP BY time
        HAVING count(player) >= nrOfAttackers) stop
WHERE start.time < stop.time AND
start.time >= (stop.time - time_limit second)
```

Løsningen som vises i listing 4.26 er ekvivalent til det ovennevnte eksempelet men presenterer en forskjell i hvordan skjemaer defineres. Ved hjelp av kommandoen *with* opprettes to midlertidige skjemaer, og innholdet og strukturen i dem er identisk til subspørringene i listing 4.25. Skjemaene kombineres deretter på den maksimale tiden mellom hver situasjon i en *where*-klausul. Denne fremgangsmåten har i tidligere seksjoner vist seg å produsere varierende resultater avhengig av hvilke attributter som inkluderes og behandles i skjemaene. Løsningen som presenteres under ligner konseptet om spørringen *Løpebane*, men i stedet for operasjoner tilknyttet spesifikke spillere, kalles funksjonen *count()* for å telle antall spillere i et gitt område.

Listing 4.26: Eksempel 2 - *Kollektivt angrep* i pseudokode

```
WITH start AS
     (SELECT time, count(player)
      FROM Match
      ...),
stop AS
     (SELECT time, count(player)
      FROM Match
      ...)
SELECT round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+ 15 sec
WHERE start.time < stop.time AND
start.time >= (stop.time - time_limit second)
```

Testing

For å sammenligne løsningene (se Vedlegg E for fullstendig kode), har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze*. Parametrene er predefinerte og presenteres under:

- **Spørring**: Kollektivt angrep

- **Kamp**: Tromsø-Tottenham
- **Omgang**: Første
- **Antall spillere i den defensive sonen**: 4
- **Antall spillere i den offensive sonen**: 3
- **Maksimal tid mellom situasjonene**: 30 sekunder
- **Område**: Defensivt område > 70, offensivt område < 30

Resultatene i tabell 4.11 viser at i konteksten av konseptet *Kollektiv forflytning*, er oppretting av skjemaer som subspørringer hensiktsmessig i søken etter en optimal løsning. Denne løsningen er mer enn 200 millisekunder raskere enn spørringen i listing 4.26. Dette viser igjen at oppretting av skjemaer via kommandoen *with* produserer varierende resultater med henblikk på kjøretid. Mangel på indeksering av attributter har tidligere blitt nevnt som en bidragsyter, men i dette tilfellet burde det ikke ha noen betydning. Grunnen til dette er at tidspunkt er det eneste attributtet som behandles i det ytre leddet av koden, og dette har ikke medført høyere kompleksitet tidligere. En mulig årsak til de lange kjøretidene kan imidlertid være at måten PostgreSQL optimaliserer og eksekverer spørringen på, øker antallet operasjoner som utføres totalt sett. Et aspekt som kan være preget av kombinasjonen mellom metoden for oppretting av skjemaer og kallet på *count*-funksjonen.

Tabell 4.11: Sammenligning av spørringene i listing 4.25 og 4.26

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.25)	11875,207 ms, 11880,475 ms 11891,938 ms, 11896,276 ms 11905,255 ms, 11921,400 ms 11922,676 ms, 11933,304 ms 11935,759 ms, 11941,148 ms	11910,3438 ms	11913,3275 ms
Eksempel 2 (4.26)	14010,401 ms, 14049,702 ms 14080,122 ms, 14098,131 ms 14105,287 ms, 14111,225 ms 14124,586 ms, 14131,484 ms 14143,224 ms, 14540,331 ms	14139,4493 ms	14108,256 ms

Løsningen (i listing 4.25) som fullførte med korteste kjøretider, benytter subspørringer til å definere skjemaer og kommandoen *where* for å kombinere dem. Spørringen under presenterer et alternativ, i form av *join*, når tabeller skal kombineres, men den bruker samme metode for å opprette skjemaer. Sammenligning mellom løsningene presenteres i tabell 4.12.

Listing 4.27: Eksempel 3 - *Kollektivt angrep* i pseudokode

```
SELECT round_time(start.time,10)-10 sec,
       round_time(stop.time,10)+ 15 sec
FROM <subquery one> start
JOIN <subquery two> stop
ON start.time < stop.time AND
start.time >= (stop.time - time_limit second)
```

Tidligere seksjoner i dette kapittelet har vist små forskjeller mellom kombineringsteknikkene *join* og *where* med henblikk på kjøretid. Hvilken teknikk som marginalt har medført korteste kjøretider har variert, et aspekt som kan ha blitt påvirket av løsningenes struktur og hvordan attributter har blitt inkludert og behandlet. Uavhengig, impliserer resultatene i dette kapittelet at de to teknikkene, i konteksten av konseptene som blitt presentert, er tilnærmet like.

Resultatene (se tabell 4.12) viser at løsningen som bruker *join* for å kombinere tabeller, er omtrent 20 millisekunder raskere i gjennomsnitt enn løsningen i listing 4.25. Spørringen i listing 4.27 er derfor implementert i analyseverktøyet.

Tabell 4.12: Sammenligning av spørringene i listing 4.25 og 4.27

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (4.25)	11875,207 ms, 11880,475 ms 11891,938 ms, 11896,276 ms 11905,255 ms, 11921,400 ms 11922,676 ms, 11933,304 ms 11935,759 ms, 11941,148 ms	11910,3438 ms	11913,3275 ms
Eksempel 2 (4.27)	11855,113 ms, 11858,190 ms 11861,268 ms, 11882,055 ms 11887,316 ms, 11889,697 ms 11892,628 ms, 11901,053 ms 11912,594 ms, 11947,043 ms	11888,6957 ms	11888,5065 ms

4.4 Videre arbeid

Spørringene som har blitt presentert i dette kapittelet, er et utvalg av mulige konsepter som Bagadus' sensor- og videosystem tilrettelegger for. Konsepter som kan undersøkes under fremtidige iterasjoner, følger under:

Akselerasjon

Informasjon om spillernes akselerasjon produseres ikke direkte av Bagadus' sensorsystem, men kan likevel genereres via operasjoner som kalkulerer hvordan attributtet *velocity* (løpshastighet) forandres over korte tidsperioder. Eksempelet under henter alle situasjoner hvor en spiller løper 4 meter per sekund raskere enn 2 sekunder tidligere. Informasjon om akselerasjon kan deretter brukes til å for eksempel plukke situasjoner hvor spillere plutselig bytter løpsretning og begynner å akselerere i motsatt retning.

```
SELECT time, player
FROM Match one, Match two
WHERE one.velocity = 4 AND
      two.velocity = 8 AND
      one.player = two.player AND
      one.time < two.time AND
      one.time > (two.time - 2 seconds)
```

Kollektiv bevegelse

Ettersom kamptabellene i Bagadus-databasen inneholder informasjon om spillernes løpshastigheter og løpsretninger, foreligger det muligheter for generering av situasjoner hvor et bestemt antall spillere løper i en bestemt retning og hastighet innenfor et avgrenset område på banen. Et eksempel i pseudokode vises under.

```
SELECT time, count(player)
FROM Match
WHERE velocity > 6 AND
      direction > 0 AND
      pos_x BETWEEN 0 AND 20 AND
      pos_y BETWEEN 20 AND 50
GROUP BY time
HAVING count(player) > 5
```

Posisjonering i mønstre

I en artikkel om Manchester Uniteds hovedtrener Louis van Gaal, skrev Bjarte Valen at 'Nederlenderen har de siste kampene gått tilbake til sin mer vante 4-3-3-formasjon og skapt taktiske trekanter som United har benyttet over hele banen.' [83]. Taktiske tilnærminger er ofte basert på hvordan spillerne skal posisjonere seg i forhold til hverandre. Dette kan danne bakgrunnen for en spørring som baserer seg på mønstre som spillere seg

i mellom danner på banen. Et eksempel (se under) kan være å finne alle situasjoner hvor posisjonene til venstreback, venstre indreløper og venstre ving danner et trekantformet mønster på banen.

```
SELECT time
FROM Match one, Match two, Match three
WHERE one.time = two.time AND
      two.time = three.time AND
      one.pos_x BETWEEN 30 AND 40 AND
      one.pos_y BETWEEN 0 AND 10 AND
      two.pos_x BETWEEN 30 AND 40 AND
      two.pos_x BETWEEN 10 AND 20 AND
      three.pos_x BETWEEN 40 AND 50 AND
      three.pos_y BETWEEN 10 AND 20 AND
      one.time = two.time AND
      two.time = three.time
```

Løp i ulike rom

For å opprettholde en dynamisk og direkte spillestil, må spillerne bevege seg i ulike rom. Dette kan danne bakgrunnen for en spørring som baserer seg på flere spilleres løpsretninger. Et eksempel kan være å finne alle situasjoner hvor én angrepsspiller løper frem og til venstre, mens en annen løper frem og til høyre.

```
SELECT time
FROM Match one, Match two
WHERE one.player = 1 AND
      two.player = 2 AND
      one.direction BETWEEN 0 AND 0.5 AND
      two.direction BETWEEN 0.6 AND 1.2 AND
      one.velocity > 6 AND
      two.velocity > 6
```

Situasjoner hvor spillere rygger

Blant spillerdataene som Bagadus' sensorsystem produserer, finnes attributtene *facing* (retning spiller står vendt mot) og *direction* (retning spiller løper mot), og disse verdiene oppgis i radianer hvor 1 radian er 57,32 grader. Dersom en spiller enten står vendt mot eller løper mot venstre på banen (fra kameraenes perspektiv), vil verdiene være mindre 0. En verdi høyrere enn 0 vil indikere motsatt retning. Dette kan danne bakgrunnen for

en spørring som finner alle situasjoner hvor en spiller løper i en annen retning enn han står vendt mot. Et eksempel i pseudokode vises under.

```
SELECT player, time
FROM Match
WHERE player = 1 AND
velocity > 6 AND
direction < 0 AND
facing > 0
```

Defensiv posisjonering

Et konsept som har blitt undersøkt, men ikke implementert, er uthenting av hendelser hvor bakerste mann holder for lang avstand til resten av forsvarsfireren. Dette er imidlertid et aspekt som er vanskelig uten kunnskap om ballens lokasjon. Grunnen til dette er at forsvarsleddet i ballbesittende situasjoner, vil posisjonere seg annerledes enn i forsvarende situasjoner. I slike tilfeller er ikke det bakerste leddet organisert til å forsvare seg. En eventuell spørring som baserer seg på avstanden mellom bakerste mann og resten av forsvarsfireren, vil derfor returnere en stor mengde resultater. En slik spørring ble undersøkt i sluttfasen av prosjektet, men implementasjon uteble grunnet omfanget av tidspunktene som ble returnert. Mulighetene for slik spørring bør imidlertid undersøkes videre.

4.5 Sammendrag

I dette kapittelet har et sett med ferdigkonstruerte spørringer blitt presentert. Disse returnerer tidspunkt i kamper hvor bestemte hendelser oppstår og er basert på fysiske spillerdata som Bagadus' sensorsystem produserer. I tilfeller hvor konsepter til spørringer har ulike fremgangsmåter, har ulike løsninger blitt implementert. Løsningene har deretter blitt sammenlignet med henblikk på kjøretid. Bakgrunnen for dette har rot i et av kravene som stilles til analyseverktøyet; at kampsituasjoner skal kunne ekstraheres på kortest mulig tid.

Når spørringer har blitt konstruert, har ulike teknikker for oppretting og kombinering av skjemaer blitt undersøkt. Med henblikk på disse teknikkene har kjøretidene variert, og mulige årsaker til dette har blitt presentert. Et stadig tilbakevendende resultat har imidlertid vært at løsninger som oppretter midlertidige og skreddersydde skjemaer, er raskere enn løsninger som bruker hele kamptabeller som skjemaer. Videre har oppretting

av skjemaer via kommandoen *with* variert mellom å være svært rask og svært treg. Kombinasjonen mellom innholdet i skjemaene og operasjonene som utføres har blitt nevnt som en årsak til dette, et aspekt som kan ha blitt påvirket av PostgreSQLs innebygde funksjoner for optimalisering. I tillegg kan mangel på indeksering av attributter i projeksjonene, ha vært et tidsøkende element ettersom skjemaene lagres midlertidig i minnet.

For å oppnå et størst mulig grunnlag for sammenligning, har løsningene blitt konstruert med relativt store parameter-rekkevidder, et aspekt som har økt kjøretidene. Med unntak av konseptene *Kollektivt forsvar* og *Kollektivt angrep* (hvis løsninger har kjøretider på mellom 10 og 15 sekunder), viser testene at spørringene returnerer resultater på maksimalt et par sekunder. Det er imidlertid realistisk å anta at trenere vil være interessert i situasjoner basert på mer konkrete argumenter. I slike tilfeller vil resultater returneres raskere. I neste kapittel presenteres spørringer som finner ulike typer statistikker.

Kapittel 5

Ekstrahering og visualisering av statistikk

I dette kapittelet presenteres ferdigkonstruerte spørringer som finner ulike typer statistikker fra kampene. Hver enkelt spørring vil presenteres med bakgrunn for spørringen, hvordan den har blitt implementert og hvordan resultatene visualiseres i brukergrensesnittet. I tilfeller hvor spørringer har ulike fremgangsmåter, har forskjellige løsninger blitt konstruert. Disse har deretter blitt sammenlignet med henblikk på kjøretid.

5.1 Testing

I tilfeller hvor spørringer har ulike løsninger, har disse blitt testet med utgangspunkt i samme retningslinjer som beskrives i seksjon 4.1.

5.2 Hjelpemetoder- og variabler

Spørringene som er konstruert i anledning uthenting av statistikker, er ikke basert på brukerdefinerte argumenter. I stedet eksekveres disse med forhåndsdefinerte argumenter når analysesidene for kampene lastes, og spørringene vil returnere informasjon om samtlige spillere. Resultatene lagres i variabler og presenteres for brukerne når popup-vinduene for de ulike statistikkene åpnes. For statistikk som angår spilleres posisjoner på banen, kalles de aktuelle metodene i kontrolleren (*MatchController.java*) to ganger; en gang for hver halvdel i kampen, og det vil være ett resultat per omgang. Omgangene skilles fra hverandre med utgangspunkt i samme metode som presenteres i seksjon 4.2.

5.3 Spørringer

5.3.1 Gjennomsnittlige posisjoner

Bakgrunn

Opta [44] samler og analyserer direkteoverførte data fra blant annet engelske fotballkamper. 'The average position widget' [45] er en av funksjonalitetene som Opta leverer og viser spillernes gjennomsnittlige posisjoner på banen når de har ballkontakt.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen til enhver tid. Inspirert av Opta, danner dette bakgrunnen for en spørring som finner hver spillers gjennomsnittlige posisjon på banen i hver omgang av en kamp.

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Gjennomsnittlige posisjoner* til databaseserveren, tar tre argumenter; navnet på kamptabellen, kampens starttidspunkt og en tekststreng som definerer hvilken omgang som skal analyseres. For å finne hver spillers gjennomsnittlige posisjon på banen i en kamp, inkluderes og behandles følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Radene som returneres ved eksekvering av spørringen hentes fra ett enkelt skjema, kamptabellen, og projeksjonen består av tre kolonner som inkluderer spilleridentifikasjon og spillerens gjennomsnittlige posisjon på henholdsvis x-aksen og y-aksen. Disse gjennomsnittene hentes ved å kalle på PostgreSQL sin innebygde funksjon *avg()* [56]. Spørringen vises i pseudokode i listing 5.1, og argumentene er markert med rødt i koden.

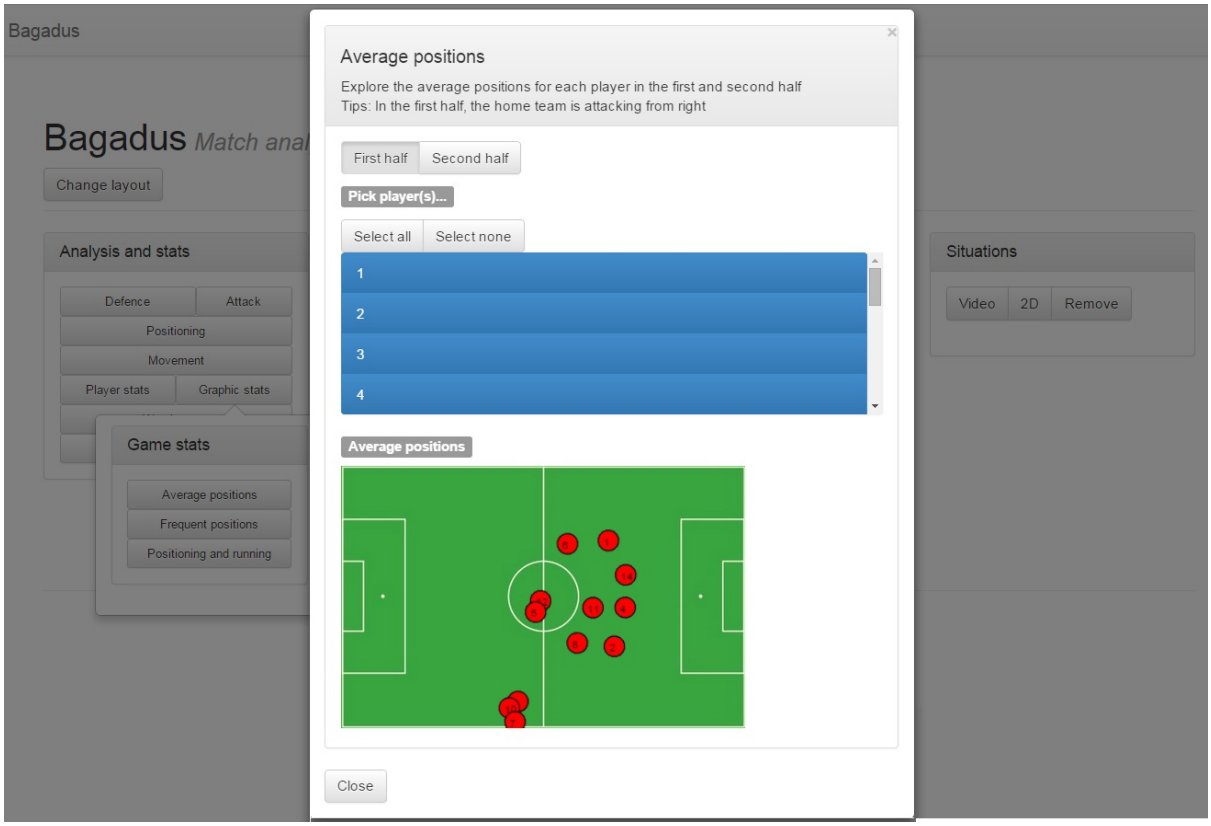
Listing 5.1: *Gjennomsnittlige posisjoner* i pseudokode

```
SELECT player, avg(pos_x), avg(pos_y)
FROM Match
WHERE time > (start_time + 60 minute)
GROUP BY player;
```

Frontend

I kampanalysesiden er spørringen representert i kategorien *Graphic Stats*. Når siden lastes, kalles den aktuelle metoden i Java-kontrolleren (*MatchController.java*) to ganger (en gang for hver omgang), og resultatene lagres i to separate lister.

Popup-vinduet som vises i figur 5.1 inneholder tre interagerbare elementer; to knapper navngitt *First half* og *Second half* og en liste med spilleridentifikasjoner. Den opptegnede fotballbanen er definert som et *canvas* [91] i HTML-koden og viser de valgte spillernes gjennomsnittlige posisjoner på banen i den valgte omgangen.



Figur 5.1: Popup for *Gjennomsnittlige posisjoner*

Tegning av objekter på et *canvas*-element, gjøres ved å kalle på dets tilhørende metode

getContext() [91]. Resultatet er et *context*-objekt som innehar en rekke funksjoner for setting av farge, text og form på objektet som tegnes. For hver spiller i listen som den ovennevnte spørringen returnerer, tegnes et rødt og sirkelformet objekt. Posisjonen til dette objektet i *canvas*-elementet, er tilsvarende den gitte spillerens gjennomsnittlige posisjon på banen i den valgte omgangen. Et eksempel i pseudokode vises i listing 5.2. Disse operasjonene gjøres hver gang bruker endrer på filtrene i popup-vinduet.

Listing 5.2: Tegning av spillere på fotballbanen

```
var field = document.getElementById('field');
var ctx =field.getContext('2d');
ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
ctx.lineJoin = ctx.lineCap = 'round';
var radius = 10;

if(half == 'first') {
  for(el in avg posisitions in first half) {
    var x = el.x*canvas.width/fieldWidth;
    var y = el.y*canvas.height/fieldHeight;
    ctx.fillStyle = 'red';
    ctx.beginPath();
    ctx.arc(x, y, radius, false, Math.PI * 2, false);
    ctx.fill();
    ctx.font = "bold " + radius + " Arial";
    ctx.fillStyle = 'black';
    ctx.fillText(el.player);
    ctx.stroke();
  }
}
```

5.3.2 Hyppige posisjoner

Bakgrunn

Spørringen som finner spillernes gjennomsnittlige posisjoner på banen, kan gi en pekepinn på om laget som et kollektiv har spilt med en angripende eller forsvarende tilnærming. Jo lavere i banen spillernes gjennomsnittsposisjoner er, jo mer defensivt har trolig taktikken vært. En viktig del av fotball er imidlertid kreativitet, og dette innebærer at spillere med

uforutsigbare egenskaper har frie tøyler i henhold til posisjonering og bevegelse.

En realistisk antakelse er at trenere ønsker informasjon om spillernes typiske bevegelser og posisjoneringsmønstre. Da kan det være hensiktsmessig å dele banen inn i soner og vise hvor ofte hver spiller befinner seg i hver sone.

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen. Spørringen som presenteres i denne seksjonen, tar utgangspunkt i 12 predefinerte soner på banen og returnerer informasjon om hvor ofte hver spiller befinner seg i hver sone.

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Hyppige posisjoner* til databaseserveren, tar tre argumenter; navnet på kamptabellen, kampens starttidspunkt og en tekststreng som definerer hvilken omgang som skal analyseres.

For å finne statistikk over hvor ofte hver spiller befinner seg i de 12 predefinerte sonene på banen, inkluderes og behandles følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Hver av løsningene som presenteres under, definerer totalt 12 skjemaer (én per sone på banen), og projeksjonene inkluderer følgende kolonner:

- En kolonne med spilleridentifikator
- 12 kolonner som teller antall rader som finnes i kamptabellen hvor den gjeldende spilleren befinner seg i den gitte sonen
- En kolonne som summerer verdiene i de 12 foregående kolonnene

Når den aktuelle kontrollermetoden mottar resultatet, omregnes verdiene fra de 12 midterste kolonnene til prosentandeler. Dette oppnås ved å dividere hver av disse verdiene

med den sammenlagte summen som finnes i den fjortende og siste kolonnen. Kontrollermetoden returnerer deretter dette resultatet til et REST-API [74] i JSON-format [34].

Radene som returneres ved eksekvering av løsningen i listing 5.3, hentes fra 12 subspørringer (en per sone på banen). I hver av disse finnes betingelser som definerer koordinatene til hver sone, og resultatet som returneres består av to kolonner i form av spilleridentifikator og antall rader i tabellen hvor spilleren befinner seg i det gitte området.

For å beregne hvor ofte en spiller befinner seg i hver sone, må informasjonen fra hver subspørring sammenlignes med summen av resultatene fra samtlige skjemaer. Dette forutsetter at resultater som inneholder 0-verdier også inkluderes. Dersom PostgreSQL-funksjonen *count()* [87] ikke finner noen forekomster for attributtet som det grupperes på, vil raden fjernes fra resultatet. Dette medfører at statistikk ikke kan beregnes for hver spiller. Subspørringene i listing 5.3 viser hvordan denne problemstillingen har blitt håndtert. Ved å legge betingelsene og kallet på *count()*-funksjonen i et eget subskjema som ikke grupperer på et attributt, vil tallet 0 returneres dersom metoden ikke finner noen forekomster. Subskjemaet og det ytre leddet av subspørringen kombineres deretter på spilleridentifikator.

Listing 5.3: Eksempel 1 - *Hyppige posisjoner* i pseudokode

```
SELECT rightAttack.player, rightAttack.pos, centerAttack.pos, ...,
      (rightAttack.pos + centerAttack.pos +...) as sum
FROM
    (SELECT a.player as player,
        (SELECT count(b.time) as pos
         FROM Match b
         WHERE b.player = a.player AND
         b.time < (start_time + 55 sec) AND
         b.pos_x BETWEEN 0 AND 26 AND
         b.pos_y BETWEEN 50 AND 69)
     FROM Match a
     GROUP BY a.player) rightAttack,
    (SELECT a.player as player,
        (SELECT count(b.time) as pos
         FROM Match b
         WHERE b.player = a.player AND
         b.time < (start_time + 55 sec) AND
```

```
            b.pos_x BETWEEN 0 AND 26 AND
            b.pos_y BETWEEN 20 AND 50)
        FROM Match a
        GROUP BY a.player) centerAttack,
        ...
WHERE rightAttack.player = centerAttack.player AND ...
```

Løsningen som vises i listing 5.4 oppretter de 12 skjemaene via kommandoen *with*. Disse skjemaene er identiske til subspørringene i den ovennevnte løsningen, og skjemaene kombineres deretter i en *where*-klausul [90]. Denne metoden for oppretting av skjemaer har ved tidligere anledninger (se kapittel 4) produsert varierende resultater, avhengig av spørringens struktur og innhold. Konseptet som presenteres i seksjon 4.3.7 om *Kollektivt forsvar/angrep* oversettes til databasekode som ligner løsningen i listing 5.4. *Where*-klausulene i skjemaene behandler liknende informasjon, og radene som returneres inkluderer resultat av kall på funksjonen *count()*. På bakgrunn av dette er mulig at løsningen under kan lide av lengre kjøretider enn løsningen i listing 5.3.

Listing 5.4: Eksempel 2 - *Hyppige posisjoner* i pseudokode

```
WITH rightAttack AS
     <subquery one>,
centerAttack AS
     <subquery two>,
...
SELECT a.player, rightAttack.pos, centerAttack.pos, ...,
      (rightAttack.pos + centerAttack.pos +...) as sum
FROM rightAttack, centerAttack, ...
WHERE rightAttack.player = centerAttack.player AND ...
```

Testing

For å sammenligne løsningene (se Vedlegg F for fullstendig kode), har hver av dem blitt eksekvert 10 ganger med kommandoen *explain analyze* [60]. Parametrene er predefinerte og presenteres under:

- **Kamp**: Tromsø-Tottenham
- **Omgang**: Første

Resultatene (se tabell 5.1) viser at 15 millisekunder skiller de gjennomsnittlige kjøretidene. Dette er i favør løsningen som oppretter skjemaer som subspørringer. Sett i forhold

til omfanget av kjøretidene er imidlertid forskjellene marginale, og det er derfor rimelig å anta at de to løsningene er relativt ekvivalente med henblikk på kompleksitet. Dette står i kontrast til testresultatene som presenteres i seksjon 4.3.7 om konseptet *Kollektivt forsvar/angrep*. Én mulig årsak til dette kan være at løsningen i listing 5.4 ikke stiller krav til resultatet av kallet på *count*-funksjonen, et tilstedeværende aspekt ved spørringen i seksjon 4.3.7.

Tabell 5.1: Sammenligning av spørringene i listing 5.3 og 5.4

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 1 (5.3)	5385,698 ms, 5399,541 ms 5403,774 ms, 5403,811 ms 5412,330 ms, 5426,374 ms 5429,737 ms, 5432,791 ms 5443,440 ms, 5446,356 ms	5418,3852 ms	5419,352 ms
Eksempel 2 (5.4)	5403,709 ms, 5410,959 ms 5420,428 ms, 5422,211 ms 5431,166 ms, 5431,301 ms 5433,540 ms, 5436,760 ms 5451,869 ms, 5490,533 ms	5433,2476 ms	5431,2335 ms

Løsningene i listing 5.3 og 5.4, kombinerer de 12 skjemaene via kommandoen *where*. Spørringene under bruker henholdsvis tilsvarende metoder for oppretting av skjemaer, men kombinerer i stedet tabellene via kommandoen *join* [62]. Resultatene vises i tabell 5.2.

Listing 5.5: Eksempel 3 - *Hyppige posisjoner* i pseudokode

```
SELECT a.player, rightAttack.pos, centerAttack.pos, ...,
      (rightAttack.pos + centerAttack.pos +...) as sum
FROM <subquery one> rightAttack
JOIN <subquery two> centerAttack
ON rightAttack.player = centerAttack.player
JOIN
...
```

Listing 5.6: Eksempel 4 - *Hyppige posisjoner* i pseudokode

```
WITH rightAttack AS
     <subquery one>,
centerAttack AS
```

```
    <subquery two>,
...
SELECT a.player, rightAttack.pos, centerAttack.pos, ...,
      (rightAttack.pos + centerAttack.pos +...) as sum
FROM rightAttack
JOIN centerAttack
ON rightAttack.player = centerAttack.player
JOIN
...
```

I kapittel 4 ble det avdekket små forskjeller mellom kombineringsteknikkene *where* og *join*, og hvilken teknikk som har utgjort ingrediensen i løsningen med korteste kjøretider, har variert. Uten direkte innsyn i og tilstrekkelig kunnskap om PostgreSQLs innebygde optimaliseringsmetoder, er det vanskelig å presentere konklusjoner om disse utfallene. Det er imidlertid tydelig at kombinasjonen mellom struktur, innhold og operasjoner i spørringer har en viss innvirkning på kompleksitet og kjøretid.

Testresultatene (se tabell 5.1 og 5.2) viser at, i konteksten av konseptet om *Hyppige posisjoner*, er kombinering av tabeller via kommandoen *join* mer hensiktsmessig enn via kommandoen *where*. Videre er resultatene også i favør oppretting av skjemaer som subspørringer. På bakgrunn av dette er derfor spørringen i listing 5.5 implementert i analyseverktøyet.

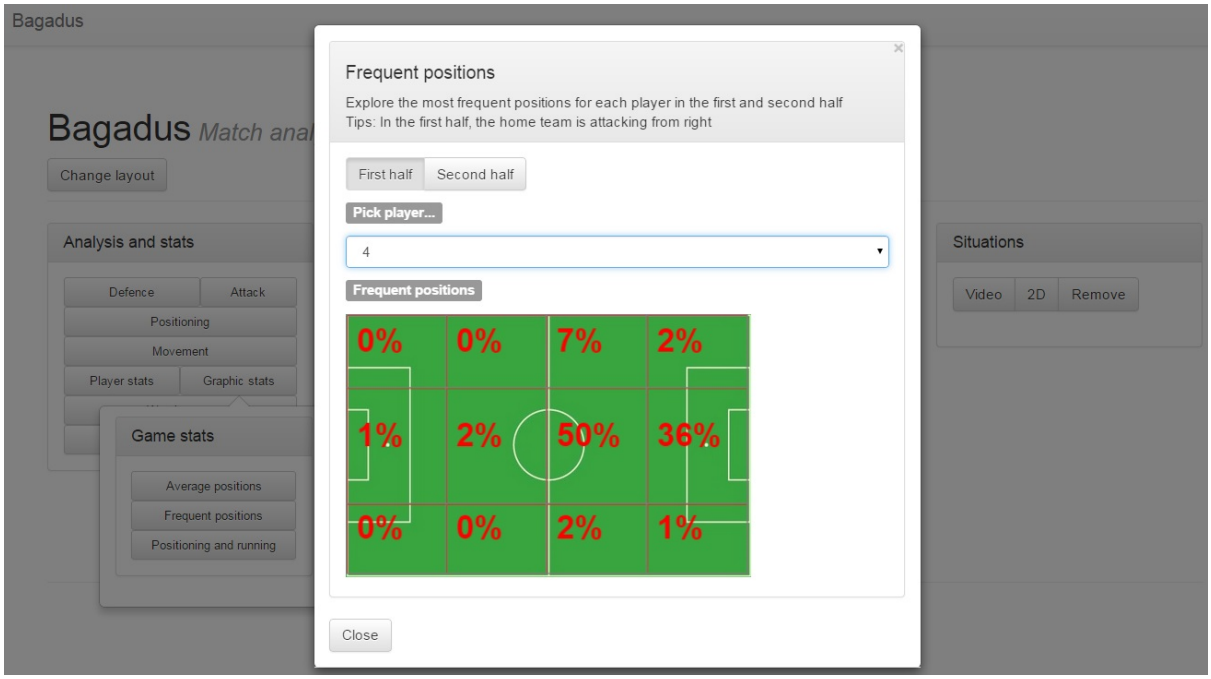
Tabell 5.2: Sammenligning av spørringene i listing 5.5 og 5.6

Spørring	Kjøretider	Gjennomsnitt	Median
Eksempel 3 (5.5)	5381,105 ms, 5384,396 ms 5386,179 ms, 5391,122 ms 5400,761 ms, 5403,408 ms 5406,040 ms, 5408,644 ms 5440,265 ms, 5454,116 ms	5405,6036 ms	5402,0845 ms
Eksempel 4 (5.6)	5397,132 ms, 5399,090 ms 5400,403 ms, 5401,751 ms 5403,844 ms, 5409,262 ms 5413,868 ms, 5416,833 ms 5428,285 ms, 5438,564 ms	5410,9032 ms	5406,553 ms

Frontend

I kampanalysesiden er spørringen *Hyppige posisjoner* representert i kategorien *Graphic stats*. Når siden lastes, kalles den aktuelle metoden i kontrolleren (*MatchController.java*) to ganger (en gang for hver omgang), og resultatene lagres i to separate lister.

Popup-vinduet som vises i figur 5.2 inneholder tre interagerbare elementer; to knapper navngitt *First half* og *Second half* og en *select*-boks [85] med spilleridentifikasjoner. Den opptegnede fotballbanen er definert som et *canvas* i HTML-koden, og de 12 predefinerte sonene visualiseres i form av en 3*4 matrise. Tegning av matrisen på den opptegnede fotballbanen gjøres på liknende måte som beskrevet for spørringen *Gjennomsnittlige posisjoner*, og posisjonen til hver rute i *canvas*-objektet, tilsvarer den tilhørende sonens posisjon på banen.

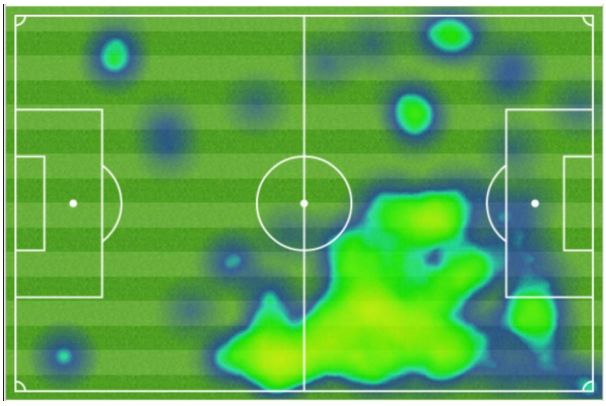


Figur 5.2: Popup for *Hyppige posisjoner*

5.3.3 Posisjonering og løpsretning

Bakgrunn

Et *heatmap* [16] er en grafisk representasjon av informasjon. I sammenheng med analysering av fotballkamper- og spillere, brukes dette for å vise hvor på banen spillerne befinner seg oftest. Sterke farger indikerer at den gjeldende spilleren besøker det fargelagte området hyppig. Et eksempel på et *heatmap* vises i figur 5.3



Figur 5.3: Heatmap [92]

Kamptabellene i Bagadus-databasen inneholder blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen
- **direction** (float) - Retning spiller løper mot. Verdien 0 er y-aksens retning
- **velocity** (float) - Spillers løpshastighet i meter per sekund

Kombinert inneholder disse attributtene informasjon om spillernes posisjoner og bevegelsesmønstre. Inspirert av konseptet om *heatmaps*, danner dette bakgrunnen for en spørring som returnerer informasjon om spillernes posisjoner, løpsretning og løpshastighet gjennom hver omgang. Denne informasjonen skal visualiseres på en opptegnet fotballbane i brukergrensesnittet.

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen *Posisjonering og løpsretning* til databaseserveren, tar fire argumenter:

- Navnet på kamptabellen
- En tekststreng som definerer hvilken omgang som skal analyseres
- En tekststreng som definerer hvilken banehalvdel som hjemmelaget angriper fra i første omgang
- Kampens starttidspunkt

For å finne statistikk over hvor spillerne posisjonerer seg og løper, inkluderes og behandles følgende attributter fra kamptabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikator
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen
- **direction** (float) - Retning spiller løper mot. Verdien 0 er y-aksens retning
- **velocity** (float) - Spillers løpshastighet i meter per sekund

Projeksjonen i den konstruerte spørringen (se listing 5.7) inneholder seks kolonner med informasjon om hver spillers posisjon, løpsretning og løpshastighet, og resultatet som returneres består av én rad per sekund i kampen. Bakgrunnen for å hente data én og ikke 20 ganger i sekundet, har rot i kompleksiteten tilknyttet prosessering av resultatet av spørringen. Den aktuelle metoden i Java-kontrolleren må traversere gjennom radene før de kan returneres i JSON-format, og frontenddelen av systemet må behandle JSON-objektene hver gang bruker vil undersøke statistikker om bestemte spillere.

Spørringen som vises i listing 5.7 definerer to skjemaer. Det ene er kamptabellen, mens det andre er en subspørring. Subspørringen runder hvert tidspunkt til nærmeste hele sekund, og skjemaene kombineres på disse tidspunktene i det ytre leddet av koden. Dette gjør at resultatet inneholder én rad med sensorinformasjon per sekund i hver omgang av kampen.

Listing 5.7: *Posisjonering og løpsretning* i pseudokode

```
SELECT a.time, a.player, a.pos_x, a.pos_y, a.direction, a.velocity
FROM Match a,
    (SELECT distinct round_time(time, 1) as time
     FROM Match) b
WHERE a.time = b.time AND
a.time < (start_time + 55 min)
```

Spørringen er løst inspirert av *heatmaps*, men skal i tillegg legge til rette for fremvisning av informasjon om spilleres løpshastighet og løpsretning. Attributtet tilknyttet løpsretning vil returneres som en tekststreng i form av enten *attacking* eller *defending* til frontend-delen av systemet, og de to retningene vil fremvises i brukergrensesnittet ved hjelp av henholdsvis blå farge og rød farge. Innholdet i tekststrengen vil settes i den aktuelle kontrollermetoden når resultatet av spørringen har blitt returnert. Kombinasjonen av følgende parametre bestemmer verdien:

- Valg av omgang
- Banehalvdelen som hjemmelaget angriper fra i første omgang

- Verdien av attributtet *direction* som returneres ved eksekvering av spørringen

Tabell 5.3 viser hvilke kombinasjoner av parametrene som resulterer i de to tekststrengene *attacking* og *defending*.

Tabell 5.3: Verdier for *retning* basert på parametrene

Tekststreng for retning	Omgang	Side i første omgang	Retning
attacking	første	venstre	>0
	andre	høyre	>0
	første	høyre	<0
	andre	venstre	<0
defending	første	høyre	>0
	andre	venstre	>0
	første	venstre	<0
	andre	høyre	<0

Frontend

I kampanalysesiden er spørringen *Posisjonering og løpsretning* representert i kategorien *Graphic stats*. Den aktuelle metoden i kontrolleren (i *MatchController.java*) kalles to ganger når analysesiden lastes, en gang for hver omgang, og resultatene lagres i to separate lister.

Informasjonen som returneres av kontrolleren inneholder hver spillers posisjon på banen, løpsretning og løpshastighet én gang i sekundet i den gjeldende omgangen. Popup-vinduet som vises i figur 5.4 presenterer følgende interagerbare elementer:

- To knapper navngitt *First half* og *Second half*
- En *select*-boks med spilleridentifikasjoner
- En *slider* med verdier for løpshastighet og en knapp som bestemmer om hastigheten skal være større eller mindre
- Tre knapper navngitt *attacking*, *defending* og *both*.

Den opptegnede fotballbanen er definert som et *canvas* i HTML-koden og viser den gjeldende spillerens posisjoner på banen i øyeblikk hvor han oppfyller kriteriene som settes i brukergrensesnittet. Disse posisjonene representeres av enten røde eller blå *dotter*, avhengig av om løpsretningen i det gitte øyeblikket er av forsvarende eller angripende

natur. Eksempelet i popup-vinduet viser posisjonene til spiller 4 i første omgang hvor han løper enten fremover eller bakover i mindre enn 10 meter per sekund.

Figur 5.4: Popup for *Posisjonering og løpsretning*

5.3.4 Todimensjonal visning av kampene

Bakgrunn

Det populære fotballstrategiske simulatorspillet *Football Manager* [31] lar bruker leve seg inn i yrket som fotballtrener, og spillet tilbyr mulighet for blant annet kjøp og salg av spillere, oppsett av taktikker og formasjoner og styring av treninger. En annen viktig funksjonalitet er to- og tredimensjonal visning av kampene som spilles. Opptegning og animasjon av fotballkamper på denne måten, forutsetter at spillernes posisjoner under kampene er kjente. Ettersom spillerposisjoner er informasjon som Bagadus' sensorsystem leverer, introduserer dette muligheter for animerte fremvisninger av kampene i Bagadus-databasen.

Kamptabellene i Bagadus-databasen inneholder verdier for blant annet følgende attributter:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikasjon
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Kombinert inneholder disse attributtene informasjon om hvor spillerne befinner seg på banen. Spørringen som presenteres i denne seksjonen, returnerer hver spillers posisjon på banen én gang i sekundet, og denne informasjonen skal brukes til å generere en todimensjonal visning av kamper.

Backend

Java-metoden (i *MatchService.java*) som konstruerer og sender spørringen til databaseserveren, tar ett argument; navnet på kamptabellen, og spørringen inkluderer og behandler følgende attributter fra tabellen:

- **timestamp** (String) - Tidspunkt og dato i lokal sentraleuropeisk tid
- **player_id** (int) - Spilleridentifikasjon
- **position** (Point) - Geografisk punkt bestående av verdier for x- og y-aksen

Spørringen som vises i listing 5.8 bruker kamptabellen og en subspørring til å definere to skjemaer. Subspørringen returnerer ett tidspunkt per sekund i kampen, og de to skjemaene kombineres på disse tidspunktene.

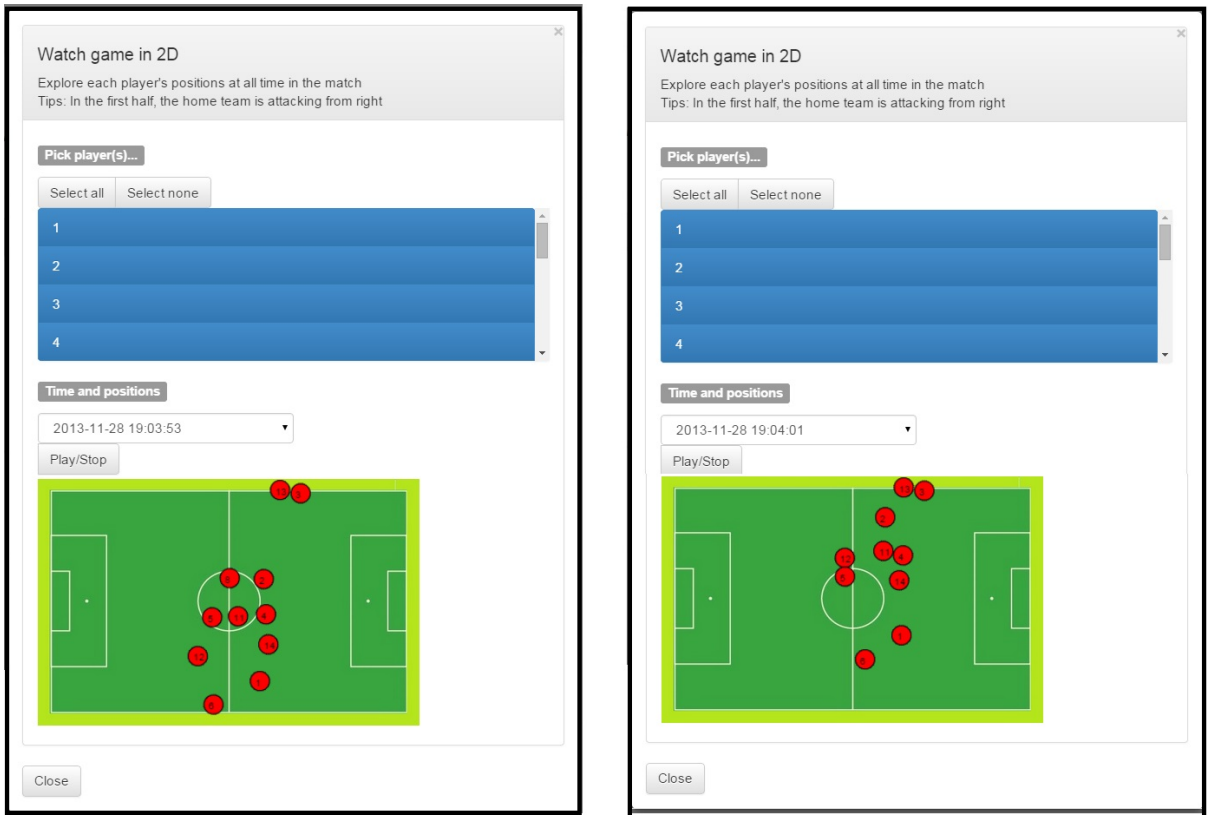
Ideen med spørringen er å tilrettelegge for grafisk fremvisning av spillernes koordinater på banen under hele kampen. For å fjerne behovet for å traverse hele listen hver gang animasjonen skal oppdateres, returnerer kontrollermetoden et *TreeMap* [48] med hvert unike tidspunkt som nøkkel. Objektene i denne datastrukturen inneholder hver spillers posisjon på banen i det øyeblikket, samt en tekststreng med det neste tidspunktet (andre kolonnen i spørringen). Dette bidrar til at grafisk fremvisning av kampen har O(n*m) i kompleksitet hvor n er antall nøkler i trestrukturen og m er antall spillere. Ytterligere forklaring presenteres under.

Listing 5.8: *Todimensjonal fremvisning av kampene* i pseudokode

```
SELECT a.time a.time+1 sec, a.player, a.pos_x, a.pos_y
FROM Match a,
    (SELECT distinct round_time(time, 1) as time
     FROM Match) b
WHERE a.time = b.time
```

Frontend

Informasjonen som behøves for generering av todimensjonal fremvisning av kamper, hentes idet kampanalysesiden lastes, og spørringen representeres av knappen *Watch game in 2D* i analysepanelet. Popup-vinduet som viser kampen inneholder en opptegnet fotballbane, samt en liste med spillere, en *select*-boks [85] med tidspunkt og en *play/stop*-knapp. Listen støtter valg av flere elementer og brukes til å definere hvilke spillere som skal animeres. Figur 5.5 viser to utdrag av europaligakampen mellom Tromsø og Tottenham fra 2013.



Figur 5.5: Popup for *2D visning av kamp*

Den opptegnede fotballbanen er definert som et *canvas* i HTML-koden, og tegning av objekter gjøres på liknende måte som tidligere beskrevet. For å kunne vise en kamp i 2D, må metoden som håndterer animering kalles fortløpende. Funksjonen som vises i listing 5.9, består av en løkke som itererer gjennom nodene i én bestemt oppføring av trestrukturen. Denne oppføringen identifiseres av den globale variabelen *currentTime*. I tillegg til informasjon om spillernes koordinater på banen, inneholder nodene en peker til det neste tidspunktet i kampen. Dette tidspunktet ugjør den nye verdien av *currentTime* når funksjonen kalles på nytt.

For at den animerte kampvisningen skal avspilles i et hensiktsmessig tempo, må en sove-funksjon kalles mellom hvert rekursive kall. Koden i listing 5.9 bruker AngularJS-funksjonen *timeout* [2]. Denne tar to parametre; navnet på funksjonen som skal forsinkes og antall millisekunder den skal forsinkes med. Ved å sette tiden til 250 millisekunder, vil den animerte kampen avspilles fire ganger så fort som den gjør i *real-time*.

Listing 5.9: AngularJS-kode som genererer todimensjonal fremvisning av kamp

```
var playInFMStyle = function() {
  var el = document.getElementById('field');
  var ctx = el.getContext('2d');
  ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);
  ctx.lineJoin = ctx.lineCap = 'round';
  var points = [], radius = 10;

  if($scope.currentTime == "" || !$scope.play) {
    return;
  }

  var map = $scope.currentTime;
  for(var i in $scope.allPositions[map]) {
    if($scope.allPosPlayersToShowContains(
       $scope.allPositions[map][i])){
      var x = $scope.allPositions[map][i].x*$scope.areaWidth/112;
      var y = (75-$scope.allPositions[map][i].y)*$scope.areaHeight/75;
      ctx.fillStyle = 'red';
      ctx.beginPath();
      ctx.arc(x, y, radius, false, Math.PI * 2, false);
      ctx.fill();
      ctx.font = "bold " + radius + " Arial";
      ctx.fillStyle = 'black';
      ctx.fillText($scope.allPositions[map][i].name,
                   x - (radius/2), y + (radius/2));
      ctx.stroke();
    }
    $scope.currentTime = $scope.allPositions[map][i].next;
  }
  poller = $timeout(playInFMStyle, 250);
};
```

```
playInFMStyle();
```

5.4 Videre arbeid

Spørringene som har blitt presentert i dette kapittelet, er et utvalg av mulige konsepter som Bagadus' sensorsystem tilrettelegger for. Mulige konsepter som kan utforskes under fremtidige iterasjoner, følger under:

Fullførte løp
Spørringen som presenteres i seksjon 4.3.3 finner alle situasjoner hvor en spesifikk spiller løper i en bestemt hastighet og retning i et bestemt antall sekunder. En utvidelse av denne kan basere seg på liknende betingelser, men i tillegg returnere informasjon om hvor løpet startet og hvor det stoppet. Denne informasjonen kan deretter visualiseres på en opptegnet fotballbane i form av piler som går fra startposisjonen til sluttposisjonen.

Et uoptimalisert og overordnet eksempel i pseudokode vises under. Den ytre spørringen henter informasjon fra tre skjemaer. Kamptabellen er definert to ganger, mens det tredje skjemaet tilsvarer spørringen som presenteres i 4.3.3. Tidspunktene som returneres av denne subspørringen, brukes deretter til å hente start- og sluttposisjonen fra henholdvis den første og den andre tabellen. Projeksjonen returnerer dermed informasjon om hvor løpet startet og hvor det sluttet.

```
SELECT x.player, x.start, x.stop, one.pos_x, one.pos_y, two,pos_x, two.
 ↪ pos_y
FROM Match one, Match two,
    (SELECT b.start as start, b.stop as stop, a.player as player
     FROM
        (SELECT time, player, velocity
         FROM Match
         WHERE velocity > 6 AND
         direction > 0 AND player = 1) a,
        (SELECT time as start, (time + 4 sec) as stop, player
         FROM Match
         WHERE velocity > 6 AND
         direction > 0 AND player = 1) b
     WHERE a.player = b.player AND
```

```
      a.time BETWEEN b.start AND b.stop
      GROUP BY b.start, b.stop, a.player
      HAVING count(a.velocity) > 20*4) x
WHERE one.time = x.start AND two.time = x.stop AND one.player = two.
  ↪ player AND two.player = x.player
```

Typiske bevegelsemønstre

Spørringen som presenteres i seksjon 5.3.2 returnerer informasjon om hvor ofte hver enkelt spiller befinner seg innenfor hver av 12 predefinerte soner på banen. En utvidelse av dette konseptet kan være en spørring som er basert på et brukerdefinert område. Spørringen kan deretter returnere informasjon om hvordan en gitt spiller typisk entrer dette området, og dette løpet kan visualiseres på en opptegnet fotballbane i form av piler.

Et eksempel i pseudokode vises under. Eksempelet tar utgangspunkt i to situasjoner og den tidsmessige avstanden mellom dem. På et tidspunkt skal den gitte spillerne befinne seg mer enn 10 meter fra det brukerdefinerte området. 5 sekunder senere skal spilleren befinne seg innenfor.

```
SELECT player, start,pos_x, start.pos_y, stop.pos_x, stop.pos_x
FROM
    (SELECT player, pos_x, pos_y, time
     FROM Match
     WHERE (pos_x NOT BETWEEN 0 AND 17 OR
     pos_y NOT BETWEEN 20 AND 50) AND
     player = 1) start,
    (SELECT player, pos_x, pos_y, time
     FROM Match
     WHERE pos_x BETWEEN 0 AND 25 AND
     pos_y BETWEEN 10 AND 60 AND
     player = 1) stop
WHERE start.time < stop.time AND
      start.time > (stop.time - 5 seconds)
```

5.5 Sammendrag

I dette kapittelet har et sett med ferdigkonstruerte spørringer som finner ulike typer statistikker, blitt presentert. Disse er ikke basert på brukerdefinerte parametre men eksekveres i stedet når kampanalysesiden i analyseverktøyet lastes. Resultatene lagres i lister

og presenteres for brukerne når popup-vinduene for spørringene åpnes.

Tilknyttet konseptet *Hyppige posisjoner*, har flere løsninger blitt konstruert. Disse løsningene har videre blitt sammenlignet med henblikk på kjøretid, hvorav den raskeste har blitt implementert i analyseverktøyet. De resterende konseptene er mindre komplekse og har derfor kun én løsning. Felles for alle spørringene, er at resultatene visualiseres på en opptegnet fotballbane i brukergrensesnittet, og inspirasjonen bak hver av dem er hentet fra blant annet spillet *Football Manager* [31] og diverse kjente fotballanalytiske konsepter.

Spørringene som har blitt beskrevet i dette og det forrige kapittelet, utgjør en viktig del av Bagadus' post-kamp analyseverktøy. Målet med implementasjonen har vært å tilby brukerne en effektiv og hurtig måte å gjennomføre kampanalyse på. Dette er en funksjonalitet som flere eksisterende systemer for fotballanalyse mangler, et aspekt som har blitt beskrevet i seksjon 2.1. I neste kapittel presenteres resultatene som har blitt oppnådd i forbindelse med denne oppgaven. Tilbakemelding fra brukerne vil beskrives, og latensene til de implementerte spørringene vil presenteres og sammenlignes med tidsbruken som eksisterende systemer krever.

Kapittel 6

Resultater

I dette kapittelet presenteres resultatene som har blitt oppnådd i forbindelse med denne oppgaven. Tilbakemelding fra brukerne vil beskrives, og latensene til de implementerte spørringene vil presenteres.

6.1 Installering og tilbakemelding fra brukere

Bagadus' post-kamp analyseverktøy er installert på Alfheim stadion i Tromsø. Systemet har blitt demonstrert for Tromsø ILs trenerteam, og det har blitt uttrykt stort engasjement for bruk under den nåværende tippeligasesongen.

> *Vår umiddelbare reaksjon er at dette kan være et stort steg frem for oss når det gjelder å finne analysedata til bruk etter kamp. Det å generere videoer i både 2D og 3D basert på punktdata, avstander mellom spillere, bevegelser (sprint osv), kollektive forflytningsdata åpner nye muligheter for oss. Det kan sies at vi lenge har diskutert hvordan vi kan få brukt ZXY- data inn i den taktiske dimensjonen av spillet, og ikke bare som rene fysiske data. Koblingen mellom ZXY og Bagadus gjør nettopp dette og med din applikasjon rundt relevante spørringer er vi i gang. Vi ser muligheter for bruk av dette i A-laget, men også i utviklingsavdelingen der vi har spillere fra 12 år og opp.*
>
> — Svein-Morten Johansen, sportssjef Tromsø IL

«*Dette er et stort steg frem fra å måtte sitte i mange timer å se gjennom videoklipp.*»

— Steinar Nilsen, hovedtrener Tromsø IL

«*Vi ønsker å kunne for eks vise spillerne hvordan de burde håndtert bestemte situasjoner annerledes, og da er det fint at video av slike situsjoner hentes så raskt*»

— Steinar Nilsen, hovedtrener Tromsø IL

«Jeg ser for meg at dette kan være nyttig for spillere ned til aldersbestemte lag. At de kan se og analysere video av seg selv i kamper. »

— Igor Aase, trener Tromsø IL

Tromsø ILs trenere er positive til å bruke systemet som et middel for utvikling av spillere og analysering av kampprestasjoner. Videre ser de også muligheter for at brukergruppen kan inkludere spillere helt ned til aldersbestemte lag. Dette forutsetter imidlertid at småguttе-, gutte- og juniorspillerne ikles sensorbelter og at video produseres av alle kampene.

Under demonstrasjonen av analyseverktøyet, ble det påpekt at systemet er en prototype og at flere iterasjoner vil være nødvendig for å gjøre det komplett. I en mail fra Tromsø ILs sportssjef, ble det formulert en rekke idéer til videre utvikling:

> *For videreutvikling har vi foreløpig følgende tanker:*
>
> 1. *Vil det være mulig etter hvert å samle opp relevante videosnutter i egne variabelmapper slik at vi enkelt kan finne frem til de videoene vi vil ha i ettertid? F.eks mappe «Bakre 4'er» ,eller «Simen Wangberg»?*
> 2. *Alternativt til punkt 1, en måte å merke av relevante videoer slik at du enkelt finner tilbake til dem?*
> 3. *Brukervennlighet. Vil det være mulig for at spillere selv kan bruke programmet for å kunne søke opp situasjoner som er relevant for dem? Og generelt brukersammenhengen? Tilgjengelighet for flere brukere?*
> 4. *I neste fase at kamera automatisk kobles til ZXY-belter og følger en spiller eksempelvis?*
>
> — Svein-Morten Johansen, sportssjef Tromsø IL

Tidspunktene som de implementerte spørringene finner, vises i en spilleliste i systemet. De lagres imidlertid ikke permanent og vil dermed fjernes dersom kampanalysesiden lastes på nytt. I den ovennevnte tilbakemeldingen fra Tromsø ILs sportssjef, ble det uttrykt et behov for lagring av spillelister i egne variabelmapper. En slik funksjonalitet vil sørge for at lagrede spillerlister når som helst kan hentes ut igjen og vises. Videre ses muligheter for at brukergruppen også kan inkludere trenere og spillere på aldersbestemte lag. I et slikt tilfelle bør fremtidige iterasjoner i større grad baseres på brukerorientert design. I konteksten av at enkeltspillere skal analysere seg selv, bør systemet ta hensyn til hvem

som bruker det og tilrettelegge for automatisk filtrering av video og statistikk basert på den gitte spilleren. En mulig løsning kan være å implementere egne analysesider for enkeltindivider som i tillegg viser hvordan deres statistikker og prestasjoner har utviklet seg over tid.

Avspilling av video i analyseverktøyet, er basert på manuell styring av Bagadus' virtuelle kamera (se seksjon 2.2.1). Videosystemet støtter imidlertid også automatisk flytting av kameraet basert på posisjonene til spillerne som *trackes* [22]. Denne funksjonaliteten bør integreres i analyseverktøyet i neste fase av prosjektet.

Til slutt har muligheter for implementering av flere spørringer, blitt diskutert med Tromsø ILs trenerteam. Spørringene som har blitt presentert i kapittel 4 og 5 er et utvalg av mulige konsepter som Bagadus tilrettelegger for, og ved fremtidige iterasjoner bør nye konsepter til spørringer utforskes og implementeres. Noen idéer har blitt presentert i seksjonene 4.4 og 5.4.

6.2 Spørringer og latenser

Kapittel 4 og 5 presenterte to sett med konstruerte spørringer. Disse returnerer henholdsvis tidspunkt hvor bestemte hendelser oppstår og statistikk, og utgjør en viktig del av Bagadus' post-kamp analyseverktøy. For å demonstrere effektiviteten til verktøyet, har hver spørring blitt eksekvert 3 ganger i systemet med forhåndsdefinerte argumenter. Disse har blitt kjørt mot databasetabellen som inneholder data fra kampen mellom Tromsø IL og Sarpsborg 08 (6. april 2015), og eksemplene under finner situasjoner som oppstår i første omgang. Statistikk som viser informasjon om spillernes posisjoneringsmønstre hentes én gang for hver omgang i kampen, og disse eksekveres sekvensielt idet kampanalysesiden lastes. Eksemplene som genererer videosammendrag presenteres under, og samtlige kjøretider vises i tabell 6.1 og 6.2.

- **Eksempel på *Spiller i område***: Alle situasjoner hvor en spesifikk spiller befinner seg innenfor høyre 16-metersboks (fra kameraenes perspektiv).
- **Eksempel på *Avstand mellom to spillere***: Alle situasjoner hvor to spesifikke spillere er mindre enn 3 meter fra hverandre innenfor høyre 16-meterboks.
- **Eksempel på *Gruppering av spillere***: Alle situasjoner hvor minst 7 spillere befinner seg innenfor høyre 16-metersboks.
- **Eksempel på *Sprinter***: Alle situasjoner hvor en spesifikk spiller løper fremover i mer enn 6 meter per sekund i minst 3 sekunder.

- **Eksempel på *Sprinter i område***: Alle situasjoner hvor en spesifikk spiller oppnår en løpshastighet på minst 7 meter per sekund innenfor venstre 16-metersboks
- **Eksempel på *Løpebane***: Alle situasjoner hvor en spesifikk spiller løper fra midtbanen og inn i venstre 16-metersboks på under 20 sekunder.
- **Eksempel på *Kollektivt angrep***: Alle situasjoner hvor 5 spillere befinner seg på høyre banehalvdel. Under 20 sekunder senere skal 3 spillere befinne seg innenfor venstre 16-metersboks

Tabell 6.1: Ekstrahering av kampsituasjoner

	Kjøringer		
Spørring	1.	2.	3.
Spiller i område	130 ms	157 ms	106 ms
Avstand mellom to spillere	133 ms	122 ms	121 ms
Gruppering av spillere	444 ms	350 ms	339 ms
Sprinter	152 ms	241 ms	171 ms
Sprinter i område	129 ms	120 ms	117 ms
Løpebane	1102 ms	1066 ms	1053 ms
Kollektivt angrep	21617 ms	22456 ms	22523 ms

Tabell 6.2: Ekstrahering av statistikker

Spørring	Kjøretider (1.omgang, 2.omgang)
Gjennomsnittlige posisjoner	1383 ms, 1274 ms 1392 ms, 1878 ms 1270 ms, 1610 ms
Hyppige posisjoner	13803 ms, 13572 ms 13270 ms, 13741 ms 13652 ms, 13597 ms
Posisjonering og løpsretning	5529 ms, 6052 ms 5246 ms, 5646 ms 5331 ms, 5739 ms
Data for todimensjonal visning av kamp	6981 ms 6790 ms 6636 ms

Kjøretidene i tabell 6.1 viser tiden det har tatt fra et søk igangsettes til resultater har blitt returnert. Spørringene som henter statistikk, eksekveres sekvensielt når kampanalysesiden lastes, og disse kjøretidene vises i tabell 6.2. Majoriteten av spørringene som

genererer videosammendrag, returnerer rader på et fåtalls hundre millisekunder. Spørringen *Kollektivt angrep* er imidlertid en del tregere, med kjøretider på rundt 20 sekunder. Videre returneres statistikker om samtlige spillere på underkant av 14 sekunder.

6.3 Konklusjon

Seksjon 2.1 beskrev en rekke eksisterende fotballanalytiske systemer. Interplay Sports [69], Longomatch [20] og Tacticalpad [13] behandler videostrømmer via manuell analysering og annotering. Systemer som StatSports [75], GPSports [25] og VX Sport [9] er mer automatiserte og levererer sensorbaserte løsninger. *Tracking* av individer som ikles disse sensorene er mulig, men systemene leverer ikke video. Det finnes derfor ingen integrering mellom fysiske spillerdata og videoopptak. Andre systemer som tilbyr automatisert innsamling av spillerdata, er Prozone [65], SportVU [39] og Match Analysis [28]. Disse bruker imidlertid kameraer til å samle informasjon.

Prozone er blant annet i stand til å måle og kalkulere spillernes posisjoner og løpshastigheter i video. Systemet leverer dermed liknende informasjon som Bagadus' sensorsystem produserer. Videre tilbyr SportVU og Match Analysis *tracking* av ball, og de er i stand til å levere informasjon basert på spillernes interaksjon med ballen. Bruk av kamera for å samle data, omtales imidlertid i [76] som upresis og ressurskrevende. Avhengig av graden av upresishet, kan dermed statistikk og videosammendrag som genereres av disse systemene, være basert på informasjon som ikke er helt nøyaktig og korrekt.

Spørringer som *Spiller i område*, *Gruppering av spillere* og til dels *Løpebane* returnerer informasjon som kan oppnås via systemer som Interplay Sports, Longomatch og Tacticalpad. Uthenting av videosammendrag i disse systemene, vil imidlertid kreve at bruker ser gjennom hele kampen og manuelt klipper ut situasjonene som oppstår. Kjøretidene som vises i tabell 6.1 utgjør dermed en brøkdel av tiden som manuell videoredigering krever. I tillegg returnerer flere av de eksekverte eksemplene informasjon som er vanskelig å oppnå manuelt. Attributter som løpshastighet, avstand mellom spillere og gjennomsnittlige posisjoner er vanskelig for det blotte øyet å kalkulere. Uthenting av slik informasjon vil i så fall forutsette grundig undersøkelse av repeterte videoavspillinger, en operasjon som kan kreve opptil flere timer.

Sensorbaserte løsninger som leveres av blant andre StatSports, GPSports og VX Sport, produserer liknende informasjon som Bagadus' sensorsystem gjør. Hvor Bagadus tilrette-

legger for automatisk uthenting av video basert på sensorinformasjon, krever de nevnte systemene manuell integrering med video fra eksterne kilder. Spillerstatistikker kan genereres automatisk, men eventuell video må redigeres manuelt, et aspekt som vil medføre en større mengde tidsbruk enn det som presenteres i tabell 6.1 og 6.2.

Oppsummert, innehar flere eksisterende systemer for fotballanalyse nyttige funksjonaliteter for gjennomføring av kampanalyse. Det er imidlertid få som integrerer videoopptak med automatisert innsamling av fysiske spillerdata. Dette kan medføre en tidkrevende prosess når bestemte kampsituasjoner skal ekstraheres. Analyseverktøyet som har blitt implementert i forbindelse med denne oppgaven, kan ses på som et bidrag til en slik problemstilling, der tid og automatisering står helt sentralt. Flere av spørringene som har blitt presentert, returnerer resultater på et fåtalls hundre millisekunder. Disse er basert på betingelser tilknyttet posisjonering av spillere, avstander mellom dem og retnings- og sonebaserte løp. Spørringer som angår forflytning av spillere er mer komplekse, hvorav den tregeste (*Kollektivt forsvar/angrep*) returnerer tidspunkt på omlag 20 sekunder. Forskjellen mellom disse tidene og tiden det tar å manuelt hente ut video av situasjoner, er dermed signifikant, og konstruksjon av spørringer har bidratt til et post-kamp analyseverktøy som muliggjør hurtig og effektiv kampanalyse.

Kapittel 7

Konklusjon

7.1 Sammendrag

I denne oppgaven har vi presentert en prototype av et post-kamp fotballanalytisk verktøy som integrerer Bagadus' sensor- og videosystem. Prototypen inneholder ferdigkonstruerte SQL-spørringer som returnerer statistikk og videosammendrag av kamprelaterte hendelser, og parametrene for spørringene defineres i verktøyets brukergrensesnitt. Motivasjonen for implementasjon ble beskrevet i første kapittel og har rot i målet om å konstruere et analysesystem som muliggjør hurtig og effektiv kampanalyse, et fraværende aspekt ved flere eksisterende analysesystemer. I tillegg har Bagadus-systemet manglet grafiske brukergrensesnitt. Ekstrahering av statistikk og videosammendrag har derfor tidligere vært en funksjonalitet utelukkende tilgjengelig for personer med informatisk bakgrunn.

Kapittel 2 innledet med en beskrivelse av andre relaterte fotballanalytiske systemer, og deres mangler og begrensninger tilknyttet tidsbruk og effektivitet ble diskutert. Deretter ble Bagadus og dets subsystemer presentert. Sensorsystemet henter informasjon fra sensorbelter (levert av ZXY Sport Tracking [82]) som spillerne har rundt midjen under kampene. Denne informasjon lagres 20 ganger i sekundet i en relasjonsdatabase og inkluderer attributter for blant annet posisjonering, løpshastighet, løpsretning og retning spiller står vendt mot. Bagadus' videosystem lagrer video av kampene som produseres av fem kameraer montert under stadiontaket på Alfheim stadion. Videoklippene er tre sekunder lange, og en hel kamp består av omtrent 1800 videofiler. Informasjonen om disse lagres i samme database som inneholder informasjon produsert av sensorsystemet. Til slutt ble Bagadus-systemets begrensninger diskutert. Dette elementet har i hovedsak angått mangel på grafiske brukergrensesnitt, et aspekt som har utgjort en viktig del av motivasjonen for utvikling av et post-kamp analyseverktøy.

I kapittel 3 ble det implementerte analyseverktøyet presentert. Verktøyet, en webapplikasjon, inneholder ferdigkonstruerte spørringer mot Bagdus-databasen som basert på brukerdefinerte parametre, returnerer statistikk og videosammendrag fra kampene. Kapittelet presenterte deretter teknologiene som har blitt benyttet, og designvalg ble redegjort for. Avspilling av genererte videosammendrag, foregår i en videoavspiller som baseres på Bagadus' virtuelle kamera, et konsept som ble beskrevet i kapittel 2.

Kapittel 4 presenterte et sett med konstruerte spørringer som returnerer tidspunkt i kamper hvor spesifikke spillere posisjonerer eller beveger seg på bestemte måter. I tilfeller hvor konsepter til spørringer har ulike fremgangsmåter, har ulike løsninger blitt konstruert og presentert. Løsningene har deretter blitt sammenlignet med henblikk på kjøretid, hvorav den mest tidseffektive er implementert i analyseverktøyet. Kapittel 5 presenterte et nytt sett med spørringer, men disse returnerer istedet statistikker over spillernes posisjonerings- og bevegelsesmønstre. Kapittelet beskrev også hvordan disse statistikkene blir visualisert i brukergrensesnittet.

Analyseverktøyet som har blitt implementert, er installert på Alfheim stadion i Tromsø. Tromsø ILs trenere har testet systemet, og tilbakemeldingene fra dem ble presentert i detalj i kapittel 6. Videre diskuterte kapittelet kjøretidene til de konstruerte spørringene. Disse ble deretter sammenlignet med tidsbruken som andre eksisterende fotballanalytiske systemer krever.

7.2 Bidrag og resultater

I denne oppgaven har vi vist hvordan Bagadus' integrerte sensor- og videosystem tilrettelegger for hurtig og effektiv kampanalyse. Dette har blitt demonstrert ved å implementere en prototype av et post-kamp analyseverktøy som inneholder ferdigkonstruerte spørringer mot Bagadus-databasen. Disse spørringene henter statistikk og videosammendrag basert på fysiske spillerdata som Bagadus' sensorsystem produserer, og de har blitt optimalisert til returnere resultater på kortest mulig tid. Flere av spørringene som genererer video av kampsituasjoner, returnerer resultater på et fåtalls hundre millisekunder. Disse er basert på konsepter for posisjonering av spillere i soner, avstander mellom dem og retnings- og sonebaserte sprinter. Spørringer som angår konkret forflytning og bevegelse av spillere er mer komplekse, og dette har preget kjøretidene. Spørringen *Løpebane* som baserer seg på situasjoner hvor enkeltindivider beveger seg fra en sone til en annen, bruker omlag et sekund på å generere resultater. Den tregeste løsningen er *Kollektivt forsvar/angrep* som bruker omtrent 20 sekunder. Statistikkene hentes sekvensielt idet analysesiden for en gitt

kamp lastes, og systemet bruker i underkant av 15 sekunder på å returnere all informasjon.

Flere av de relaterte systemene som ble presentert i seksjon 2.1, krever at bruker ser gjennom hele kamper og manuelt klipper ut situasjoner av interesse. Andre systemer gjør forsøk på automatisering men er ofte preget av upresise og ressurskrevende teknologier. Vi har vist at Bagadus' post-kamp analyseverktøy tilbyr funksjonalitet som er hensiktsmessig for gjennomføring av hurtig og effektiv kampanalyse. Spesifikke statistikker og kampsituasjoner kan genereres, og i verste fall tar det sekunder før resultater returneres.

Analyseverktøyet er installert på Alfheim stadion i Tromsø og har blitt demonstrert for Tromsø ILs trenere. Brukerne har uttrykt stort engasjement og er positive til å bruke verktøyet til å analyse kampprestasjoner. I tillegg ser de bruksmuligheter for spillere på Tromsø ILs aldersbestemte lag. Til slutt har vi også inngått et samarbeid med det norske herrelandslaget i fotball, og en demo av prototypen ble presentert for landslagssjef Per Mathias Høgmo under et seminar på Ullevål stadion høsten 2014.

Figur 7.1: Tromsø ILs hovedtrener Steinar Nilsen tester Bagadus' analyseverktøy

7.3 Videre arbeid

Analyseverktøyet er en prototype, og flere iterasjoner vil være nødvendig for å gjøre det komplett. I denne oppgaven har fokuset vært på funksjonalitet (via spørringer) fremfor design. Ved videre utvikling, bør brukerne spille en større rolle i designprosessen.

I kapittel 6 presenterte vi tilbakemeldingene som Tromsø ILs trenere ga i etterkant av demonstrasjonen av analyseverktøyet. En funksjonalitet som ble etterspurt var permanent lagring av spillelister. I den nåværende versjonen av systemet, lagres spørringsresultatene midlertidig, og de vil dermed fjernes dersom kampanalysesiden lastes på nytt. Ved videre utvikling bør bruker kunne lagre resultater i egne variabelmapper slik at de lett kan søkes opp og hentes ut. Et annet aspekt var tilrettelegging for ulike typer brukere. I konteksten

av at en spiller skal analysere seg selv, bør analyseverktøyet kunne vise hvordan denne spillerens prestasjoner har utviklet seg over tid. I den anledning kan det være hensiktsmessig å implementere egne analysesider for enkeltindivider.

Når videosammendrag skal vises, spilles disse i en avspiller som er basert på Bagadus' virtuelle kamera. Ulike knapper og *slidere* kan deretter brukes til å flytte på kameraet. Videosystemet støtter imidlertid også automatisk flytting av kameraet basert på posisjonene til spillere som skal *trackes*. Denne avspillingsfunksjonen bør integreres i analyseverktøyet i neste fase av prosjektet.

Mulighetene som Bagadus' sensor- og videosystem legger til rette for, er ikke begrenset til spørringene som har blitt implementert i analyseverktøyet. I kapittel 4 og 5 har det blitt foreslått mulige konsepter for spørringer som kan undersøkes videre. Eksempler er situasjoner tilknyttet spilleres akselerasjon, kollektive bevegelser hvor et bestemt antall spillere løper i en bestemt hastighet, posisjonering i mønstre, spillere som løper i ulike retninger og situasjoner hvor spillere løper i en annen retning enn de står vendt mot. I neste fase av prosjektet bør ytterligere konsepter for spørringer undersøkes.

Vedlegg A

Listing 1: Eksempel 1 - *Spiller i område*

```
EXPLAIN ANALYZE
SELECT distinct tag_id,
       (timestamp+interval '-10' second) as start,
       (timestamp+interval '+10' second) as end
FROM
       (SELECT tag_id as tag_id,
               round_timestamp(timestamp, 10) as timestamp
       FROM match_tromso_tottenham_2013_uefa_r13
       WHERE (position[0] between 0.525 and 17.0625) AND
       (position[1] BETWEEN 14.812827763496145 and 55.64953727506427)
       AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)
       AND (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
       tag_id = 5 or tag_id = 6 or tag_id= 7 or
       tag_id = 8 or tag_id = 9 or tag_id = 10 or
       tag_id = 11 or tag_id = 12 or
       tag_id = 13 or tag_id = 14)) a
order by tag_id, (timestamp+interval '-10' second
);
```

Listing 2: Eksempel 2 - *Sprinter i område*

```
  EXPLAIN ANALYZE
  select distinct tag_id,
         (timestamp+interval '-10' second) as start,
         (timestamp+interval '+10' second) as end
  FROM

  (SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
  FROM match_tromso_tottenham_2013_uefa_r13
```

```
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427 AND
'2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 1
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 2
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 3
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 4
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 5
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
```

```
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 6
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 7
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 8
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 9
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 10
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 11
```

```
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 12
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 13
UNION
SELECT tag_id as tag_id, round_timestamp(timestamp, 10) as timestamp
FROM match_tromso_tottenham_2013_uefa_r13
where position[0] between 0.525 and 17.0625 and
position[1] between 14.812827763496145 and 55.64953727506427
AND '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
tag_id = 14
) a
order by tag_id, (timestamp+interval '-10' second);
```

Vedlegg B

Listing 3: Eksempel 1 - *Avstand mellom to spillere*

```
EXPLAIN ANALYZE
select distinct tag_one, tag_two,
      (timestamp+interval '-10' second) as start,
      (timestamp+interval '+10' second) as end
from
(select a.tag_id as tag_one, b.tag_id as tag_two,
        round_timestamp(a.timestamp, 10) as timestamp
from match_tromso_tottenham_2013_uefa_r13 a,
     match_tromso_tottenham_2013_uefa_r13 b
where a.tag_id != b.tag_id and a.timestamp = b.timestamp and
and a.position[0] between 0 and 105
and a.position[1] between 0 and 68
and b.position[0] between 0 and 105
and b.position[1] between 0 and 68
AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
and a.tag_id = 1 AND b.tag_id = 4
and a.position IN circle(b.position, 5)) a
ORDER BY tag_one, tag_two, (timestamp+interval '-10' second);
```

Listing 4: Eksempel 2 - *Avstand mellom to spillere*

```
EXPLAIN ANALYZE
WITH player_one as
    (SELECT tag_id, timestamp, position
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE tag_id = 1 AND
     position[0] BETWEEN 0 and 105 AND
     position[1] between 0 and 68 AND
    '2013-11-28 19:03:51'> (timestamp +interval '-55' minute)),
```

```
player_two as
    (SELECT tag_id, timestamp, position
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE tag_id = 4 AND
     position[0] BETWEEN 0 and 105 AND
     position[1] between 0 and 68 AND
     '2013-11-28 19:03:51'> (timestamp +interval '-55' minute))

SELECT distinct tag_one, tag_two,
      (timestamp+interval '-10' second) as start,
      (timestamp+interval '+10' second)
FROM
(SELECT distinct player_one.tag_id as tag_one,
        player_two.tag_id as tag_two,
        round_timestamp(player_one.timestamp,10) as timestamp
FROM player_one, player_two
WHERE player_one.timestamp = player_two.timestamp AND
      player_one.position IN circle(player_two.position, 5)) a
ORDER BY tag_one, tag_two, (timestamp+interval '-10' second) x;
```

Listing 5: Eksempel 3 - *Avstand mellom to spillere*

```
EXPLAIN ANALYZE
SELECT distinct tag_one, tag_two,
      (timestamp+interval '-10' second) as start,
      (timestamp+interval '+10' second)
FROM
(SELECT distinct one.tag_id as tag_one, two.tag_id as tag_two,
        round_timestamp(one.timestamp,10) as timestamp
FROM

(SELECT tag_id, timestamp, position
 FROM match_tromso_tottenham_2013_uefa_r13
 WHERE tag_id = 1 AND
 position[0] BETWEEN 0 and 105 AND
 position[1] between 0 and 68 AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) one,
```

```
(SELECT tag_id, timestamp, position
 FROM match_tromso_tottenham_2013_uefa_r13
 WHERE tag_id = 4 AND
 position[0] BETWEEN 0 and 105 AND
 position[1] between 0 and 68 AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) two

WHERE one.timestamp = two.timestamp AND
one.position IN circle(two.position, 5)) a
ORDER BY tag_one, (timestamp+interval '-10' second);
```

Listing 6: Eksempel 4 - *Avstand mellom to spillere*

```
EXPLAIN ANALYZE
SELECT distinct one, two, (timestamp+interval '-10' second) as start,
      (timestamp+interval '+10' second)
from
(select a.tag_id as one, b.tag_id as two,
        round_timestamp(a.timestamp,10) as timestamp
 from match_tromso_tottenham_2013_uefa_r13 a
 join match_tromso_tottenham_2013_uefa_r13 b
 on a.position IN circle(b.position, 5)
 AND a.timestamp = b.timestamp
 where a.position[0] between 0 and 105
 and a.position[1] between 0 and 68
 and b.position[0] between 0 and 105
 and b.position[1] between 0 and 68
 and '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
 and a.tag_id = 1 AND b.tag_id = 4) a
 ORDER BY one, (timestamp+interval '-10' second);
```

Listing 7: Eksempel 5 - *Avstand mellom to spillere*

```
EXPLAIN ANALYZE
SELECT distinct tag_one, tag_two,
      (timestamp+interval '-10' second) as start,
      (timestamp+interval '+10' second)
FROM
(SELECT distinct one.tag_id as tag_one, two.tag_id as tag_two,
```

```
        round_timestamp(one.timestamp,10) as timestamp
FROM
(SELECT tag_id, timestamp, position
 FROM match_tromso_tottenham_2013_uefa_r13
 WHERE tag_id = 1 AND
 position[0] BETWEEN 0 and 105 AND
 position[1] between 0 and 68 AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) one
JOIN
(SELECT tag_id, timestamp, position
 FROM match_tromso_tottenham_2013_uefa_r13
 WHERE tag_id = 4 AND
 position[0] BETWEEN 0 and 105 AND
 position[1] between 0 and 68 AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) two
 ON one.timestamp = two.timestamp AND
 one.position IN circle(two.position, 5)) a
ORDER BY tag_one, (timestamp+interval '-10' second);
```

Vedlegg C

Listing 8: Eksempel 1 - *Sprinter*

```
EXPLAIN ANALYZE
select distinct tag_id, start, stop
from
(select a.tag_id,
       (round_timestamp(b.start, 10)+interval '-10' second) as start,
       (round_timestamp(b.end,10) +interval '+10' second) as stop,
        count(a.vel)
from
    (select tag_id, timestamp as start, velocity as vel
     from match_tromso_tottenham_2013_uefa_r13
     where velocity >= 5
     AND direction  < 0 AND
     '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)
     and (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
     tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
     tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
     tag_id = 13 or tag_id = 14)
     group by tag_id, timestamp, velocity) a,
    (select tag_id, timestamp as start,
            (timestamp + interval '+3' second) as end
     from match_tromso_tottenham_2013_uefa_r13
     where velocity >= 5 AND
     direction  < 0 and (tag_id = 1 or tag_id = 2 or tag_id = 3 or
     tag_id = 4 or tag_id = 5 or tag_id = 6 or tag_id = 7 or
     tag_id = 8 or tag_id = 9 or tag_id = 10 or tag_id = 11 or
     tag_id = 12 or tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) b
where a.start between b.start and b.end AND a.tag_id = b.tag_id
```

```
group by a.tag_id, b.start, b.end
having count(a.vel) > (20*3))main
ORDER BY tag_id, start;
```

Listing 9: Eksempel 2 - *Sprinter*

```
with a as
(select tag_id, timestamp as start, velocity as vel
 from match_tromso_tottenham_2013_uefa_r13
 where velocity >= 5
 AND direction  < 0 AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
 (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
 tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
 tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
 tag_id = 13 or tag_id = 14)),
b as
(select tag_id, timestamp as start,
        (timestamp + interval '+3' second) as end
 from match_tromso_tottenham_2013_uefa_r13
 where velocity >= 5 AND direction  < 0 and
 (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
 tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
 tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
 tag_id = 13 or tag_id = 14) AND
 '2013-11-28 19:03:51' > (timestamp +interval '-55' minute))

select distinct tag_id, start, stop
from
(select a.tag_id,
       (round_timestamp(b.start, 10)+interval '-10' second)as start,
       (round_timestamp(b.end,10) +interval '+10' second) as stop,
        count(a.vel)
from a, b
where a.start between b.start and b.end AND a.tag_id = b.tag_id
group by a.tag_id, b.start, b.end
having count(a.vel) > (20*3))main
ORDER BY tag_id, start;
```

Listing 10: Eksempel 3 - *Sprinter*

```
EXPLAIN ANALYZE
select distinct tag_id, start, stop
from
(select a.tag_id,
       (round_timestamp(b.start, 10)+interval '-10' second) as start,
       (round_timestamp(b.end,10) +interval '+10' second) as stop,
        count(a.velocity)
from match_tromso_tottenham_2013_uefa_r13 a,
    (select tag_id, timestamp as start,
     (timestamp + interval '+3' second) as end
     from match_tromso_tottenham_2013_uefa_r13
     where velocity >= 5 AND direction  < 0 and
     (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
     tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
     tag_id = 9 or tag_id = 10 or tag_id = 11 or
     tag_id = 12 or tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) b
 where a.timestamp between b.start and b.end AND
 a.tag_id = b.tag_id AND
 a.velocity >= 5 AND a.direction < 0
 group by a.tag_id, b.start, b.end
 having count(a.velocity) > (20*3))main
ORDER BY tag_id, start;
```

Listing 11: Eksempel 4 - *Sprinter*

```
EXPLAIN ANALYZE
select distinct tag_id, start, stop
from
(select a.tag_id,
        (round_timestamp(b.start, 10)+interval '-10' second) as start,
        (round_timestamp(b.end,10) +interval '+10' second) as stop,
        count(a.vel)
from
  (select tag_id, timestamp as start, velocity as vel
   from match_tromso_tottenham_2013_uefa_r13
   where velocity >= 5
```

```
   AND direction  < 0 AND
   '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
   (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
   tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
   tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
   tag_id = 13 or tag_id = 14)
   group by tag_id, timestamp, velocity) a
JOIN
  (select tag_id, timestamp as start,
          (timestamp + interval '+3' second) as end
   from match_tromso_tottenham_2013_uefa_r13
   where velocity >= 5 AND direction  < 0 and
   (tag_id = 1 or tag_id = 2 or tag_id = 3 or
   tag_id = 4 or tag_id = 5 or tag_id = 6 or
   tag_id = 7 or tag_id = 8 or tag_id = 9 or
   tag_id = 10 or tag_id = 11 or tag_id = 12 or
   tag_id = 13 or tag_id = 14) AND
   '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) b
ON a.start between b.start and b.end AND a.tag_id = b.tag_id
group by a.tag_id, b.start, b.end
having count(a.vel) > (20*3))main
ORDER BY tag_id, start;
```

Listing 12: Eksempel 5 - *Sprinter*

```
EXPLAIN ANALYZE
with a as
  (select tag_id, timestamp as start, velocity as vel
   from match_tromso_tottenham_2013_uefa_r13
   where velocity >= 5 and
   direction  < 0 AND
   '2013-11-28 19:03:51' > (timestamp +interval '-55' minute) and
   (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
    tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
    tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
    tag_id = 13 or tag_id = 14)),
b as
  (select tag_id, timestamp as start,
```

```
        (timestamp + interval '+3' second) as end
   from match_tromso_tottenham_2013_uefa_r13
   where velocity >= 5 AND direction  < 0 and
   (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
   tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
   tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
   tag_id = 13 or tag_id = 14) AND
   '2013-11-28 19:03:51' > (timestamp +interval '-55' minute))

select distinct tag_id, start, stop
from
(select a.tag_id,
       (round_timestamp(b.start, 10)+interval '-10' second) as start,
       (round_timestamp(b.end,10) +interval '+10' second) as stop,
        count(a.vel)
 from a
 JOIN b
 ON a.start between b.start and b.end AND a.tag_id = b.tag_id
 group by a.tag_id, b.start, b.end
 having count(a.vel) > (20*3))main
ORDER BY tag_id, start;
```

Vedlegg D

Listing 13: Eksempel 1 - *Løpebane*

```
EXPLAIN ANALYZE
SELECT distinct tag_id, (start_time + interval '-10' second) as start,
       (stop_time + interval '+10' second) as stop
FROM
 (SELECT start.tag_id as tag_id,
         round_timestamp(start.timestamp, 10) as start_time,
         round_timestamp(stop.timestamp,10) as stop_time
  FROM
  (SELECT tag_id,  timestamp
   FROM match_tromso_tottenham_2013_uefa_r13
   WHERE position[0] between 27.15625 AND 37.15625 AND
   position[1] between 35.7 AND 45.7 AND
   (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
   tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
   tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
   tag_id = 13 or tag_id = 14) AND
   '2013-11-28 19:03:51'> (timestamp +interval '-55' minute)) start,
 (SELECT tag_id, timestamp
  FROM match_tromso_tottenham_2013_uefa_r13
  WHERE
  position[0] between 3.53125 AND 13.53125 AND
  position[1] between 29.6625 AND 39.6625 AND
  (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
  tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
  tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
  tag_id = 13 or tag_id = 14) AND
  '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) stop
 WHERE start.tag_id = stop.tag_id AND
```

```
 stop.timestamp < (start.timestamp + interval '14' second)
 AND stop.timestamp > start.timestamp) x
ORDER BY tag_id, (start_time + interval '-10' second);
```

Listing 14: Eksempel 2 - *Løpebane*

```
EXPLAIN ANALYZE
with start as
    (SELECT tag_id, timestamp
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE
     position[0] between 27.15625 AND 37.15625 AND
     position[1] between 35.7 AND 45.7 AND
     (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
     tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
     tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
     tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51'> (timestamp +interval '-55' minute)),
stop as
    (SELECT tag_id, timestamp
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE
     position[0] between 3.53125 AND 13.53125 AND
     position[1] between 29.6625 AND 39.6625 AND
     (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
     tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
     tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
     tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51' > (timestamp +interval '-55' minute))

SELECT distinct tag_id, (start_time + interval '-10' second) as start,
      (stop_time + interval '+10' second) as stop
FROM
  (SELECT start.tag_id as tag_id,
          round_timestamp(start.timestamp, 10) as start_time,
          round_timestamp(stop.timestamp,10) as stop_time
   FROM start, stop
   WHERE start.tag_id = stop.tag_id AND
```

```
   stop.timestamp < (start.timestamp + interval '14' second)
   AND stop.timestamp > start.timestamp) x
 ORDER BY tag_id, (start_time + interval '-10' second);
```

Listing 15: Eksempel 3 - *Løpebane*

```
EXPLAIN ANALYZE
SELECT distinct start.tag_id,
       (round_timestamp(start.timestamp,10) + interval '-10' second),
       (round_timestamp(stop.timestamp,10) + interval '+10' second)
FROM match_tromso_tottenham_2013_uefa_r13 start,
     match_tromso_tottenham_2013_uefa_r13 stop
WHERE start.position[0] between 27.15625 AND 37.15625 AND
      start.position[1] between 35.7 AND 45.7 AND
      (start.tag_id = 1 or start.tag_id = 2 or start.tag_id = 3 or
      start.tag_id = 4 or start.tag_id = 5 or start.tag_id = 6 or
      start.tag_id = 7 or start.tag_id = 8 or start.tag_id = 9 or
      start.tag_id = 10 or start.tag_id = 11 or start.tag_id = 12 or
      start.tag_id = 13 or start.tag_id = 14) AND
      start.tag_id = stop.tag_id AND
      stop.position[0] between 3.53125 AND 13.53125 AND
      stop.position[1] between 29.6625 AND 39.6625 AND
      stop.timestamp < (start.timestamp + interval '14' second) AND
      stop.timestamp > start.timestamp AND
      '2013-11-28 19:03:51' > (start.timestamp +interval '-55' minute)
ORDER BY tag_id,
        (round_timestamp(start.timestamp,10) + interval '-10' second);
```

Listing 16: Eksempel 4 - *Løpebane*

```
EXPLAIN ANALYZE
SELECT distinct tag_id, (start_time + interval '-10' second) as start,
       (stop_time + interval '+10' second) as stop
FROM
 (SELECT start.tag_id as tag_id,
        round_timestamp(start.timestamp, 10) as start_time,
        round_timestamp(stop.timestamp,10) as stop_time
  FROM
   (SELECT tag_id, timestamp
```

```
    FROM match_tromso_tottenham_2013_uefa_r13
    WHERE
    position[0] between 27.15625 AND 37.15625 AND
    position[1] between 35.7 AND 45.7 AND
    (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
    tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
    tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
    tag_id = 13 or tag_id = 14) AND
    '2013-11-28 19:03:51'> (timestamp +interval '-55' minute)) start
 JOIN
   (SELECT tag_id, timestamp
    FROM match_tromso_tottenham_2013_uefa_r13
    WHERE
    position[0] between 3.53125 AND 13.53125 AND
    position[1] between 29.6625 AND 39.6625 AND
    (tag_id = 1 or tag_id = 2 or tag_id = 3 or tag_id = 4 or
    tag_id = 5 or tag_id = 6 or tag_id = 7 or tag_id = 8 or
    tag_id = 9 or tag_id = 10 or tag_id = 11 or tag_id = 12 or
    tag_id = 13 or tag_id = 14) AND
    '2013-11-28 19:03:51' > (timestamp +interval '-55' minute)) stop
 ON start.tag_id = stop.tag_id AND
    stop.timestamp < (start.timestamp + interval '14' second)
    AND stop.timestamp > start.timestamp) x
ORDER BY tag_id, (start_time + interval '-10' second)
```

Listing 17: Eksempel 5 - *Løpebane*

```
EXPLAIN ANALYZE
with start as
    (SELECT distinct tag_id,  timestamp
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE
     position[0] between 27.15625 AND 37.15625 AND
     position[1] between 35.7 AND 45.7 AND
     (tag_id = 1 or tag_id = 2 or tag_id = 3 or
     tag_id = 4 or tag_id = 5 or tag_id = 6 or
     tag_id = 7 or tag_id = 8 or tag_id = 9 or
     tag_id = 10 or tag_id = 11 or tag_id = 12 or
```

```
     tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51'> (timestamp +interval '-55' minute)),
stop as
    (SELECT distinct tag_id, timestamp
     FROM match_tromso_tottenham_2013_uefa_r13
     WHERE
     position[0] between 3.53125 AND 13.53125 AND
     position[1] between 29.6625 AND 39.6625 AND
     (tag_id = 1 or tag_id = 2 or tag_id = 3 or
     tag_id = 4 or tag_id = 5 or tag_id = 6 or
     tag_id = 7 or tag_id = 8 or tag_id = 9 or
     tag_id = 10 or tag_id = 11 or tag_id = 12 or
     tag_id = 13 or tag_id = 14) AND
     '2013-11-28 19:03:51' > (timestamp +interval '-55' minute))

SELECT distinct tag_id,
       (start_time + interval '-10' second) as start,
       (stop_time + interval '+10' second) as stop
FROM
(SELECT start.tag_id as tag_id,
        round_timestamp(start.timestamp, 10) as start_time,
        round_timestamp(stop.timestamp,10) as stop_time
FROM start
JOIN stop
ON start.tag_id = stop.tag_id AND
   stop.timestamp < (start.timestamp + interval '14' second)
AND stop.timestamp > start.timestamp) x
ORDER BY tag_id,
         (start_time + interval '-10' second);
```

Vedlegg E

Listing 18: Eksempel 1 - *Kollektivt angrep*

```
EXPLAIN ANALYZE
SELECT distinct (round_timestamp(counter_start.timestamp, 10)
                  + interval '-10'second) as start,
                (round_timestamp(counter_end.timestamp, 10)
                  + interval '15' second) as stop
FROM
 (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
  FROM match_tromso_tottenham_2013_uefa_r13 a
  WHERE a.position[0] > 70 AND a.position[1] between 0 AND 68
  AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
  GROUP BY a.timestamp
  HAVING count(tag_id) >=4) counter_start,
 (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
  FROM match_tromso_tottenham_2013_uefa_r13 a
  WHERE a.position[0] < 30 AND a.position[1] between 0 AND 68
  AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
  GROUP BY a.timestamp
  HAVING count(tag_id) >= 3) counter_end

WHERE counter_start.timestamp < counter_end.timestamp AND
      counter_start.timestamp >=
       (counter_end.timestamp + interval '-30' second)
ORDER BY (round_timestamp(counter_start.timestamp, 10)
          + interval '-10' second);
```

Listing 19: Eksempel 2 - *Kollektivt angrep*

```
EXPLAIN ANALYZE
with counter_start as
```

```
    (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
     FROM match_tromso_tottenham_2013_uefa_r13 a
     WHERE a.position[0] > 70 AND a.position[1] between 0 AND 68
     AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
     GROUP BY a.timestamp
     HAVING count(tag_id) >=4),
counter_end as
    (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
     FROM match_tromso_tottenham_2013_uefa_r13 a
     WHERE a.position[0] < 30 AND a.position[1] between 0 AND 68
     AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
     GROUP BY a.timestamp
     HAVING count(tag_id) >= 3)

SELECT distinct (round_timestamp(counter_start.timestamp, 10)
                 + interval '-10'second) as start,
                (round_timestamp(counter_end.timestamp, 10)
                 + interval '15' second) as stop
FROM counter_start, counter_end
WHERE counter_start.timestamp < counter_end.timestamp AND
      counter_start.timestamp >=
         (counter_end.timestamp + interval '-30' second)
ORDER BY (round_timestamp(counter_start.timestamp, 10)
           + interval '-10' second)
```

Listing 20: Eksempel 3 - *Kollektivt angrep*

```
EXPLAIN ANALYZE
SELECT distinct (round_timestamp(counter_start.timestamp, 10)
                  + interval '-10'second) as start,
                (round_timestamp(counter_end.timestamp, 10)
                  + interval '15' second) as stop
FROM
  (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
   FROM match_tromso_tottenham_2013_uefa_r13 a
   WHERE a.position[0] > 70 AND a.position[1] between 0 AND 68
   AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
   GROUP BY a.timestamp
```

```
   HAVING count(tag_id) >=4) counter_start
JOIN
  (SELECT a.timestamp as timestamp, count(a.tag_id) as cnt
   FROM match_tromso_tottenham_2013_uefa_r13 a
   WHERE a.position[0] < 30 AND a.position[1] between 0 AND 68
   AND '2013-11-28 19:03:51' > (a.timestamp +interval '-55' minute)
   GROUP BY a.timestamp
   HAVING count(tag_id) >= 3) counter_end
ON counter_start.timestamp < counter_end.timestamp AND
   counter_start.timestamp >=
     (counter_end.timestamp + interval '-30' second)
ORDER BY (round_timestamp(counter_start.timestamp, 10)
          + interval '-10' second)
```

Vedlegg F

Listing 21: Eksempel 1 - *Hyppige posisjoner*

```
EXPLAIN ANALYZE
SELECT distinct one.tag_id as tag_id, one.rightAttack,
   two.centerAttack, three.leftAttack, four.rightAttackMid,
   five.centerAttackMid, six.leftAttackMid, seven.rightDefMid,
   eight.centerDefMid, nine.leftDefMid,
   ten.rightDef, eleven.centerDef, twelwe.leftDef,
   (one.rightAttack + two.centerAttack +three.leftAttack +
   four.rightAttackMid+five.centerAttackMid + six.leftAttackMid +
   seven.rightDefMid + eight.centerDefMid + nine.leftDefMid +
   ten.rightDef + eleven.centerDef + twelwe.leftDef) as sum
FROM
(select a.tag_id as tag_id,
    (select count(b.timestamp) as rightAttack
     from match_tromso_tottenham_2013_uefa_r13 b
     where a.tag_id = b.tag_id AND
     b.position[0] between 0 and 26 AND
     '2013-11-28 19:03:51' >(b.timestamp+interval '-55' minute) AND
     b.position[1] between 50 and 69)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by a.tag_id) one,

(select a.tag_id as tag_id,
    (select count(b.timestamp) as centerAttack
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 0 and 26 AND
     '2013-11-28 19:03:51' >(b.timestamp+interval '-55' minute) AND
     b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
```

```
 group by tag_id) two,

(select  a.tag_id as tag_id,
    (select count(b.timestamp) as leftAttack
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 0 and 26 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) three,

(select a.tag_id as tag_id,
    (select count(b.timestamp) as rightAttackMid
     from match_tromso_tottenham_2013_uefa_r13 b
     where b.position[0] between 26 and 52 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 50 and 69 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) four,

(select a.tag_id as tag_id,
    (select count(b.timestamp) as centerAttackMid
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 26 and 52 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) five,

 (select distinct a.tag_id as tag_id,
    (select count(b.timestamp) as leftAttackMid
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 26 and 52 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) six,
```

```
 (select distinct a.tag_id as tag_id,
    (select count(b.timestamp) as rightDefMid
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) seven,

 (select distinct a.tag_id as tag_id,
    (select count(b.timestamp) as centerDefMid
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 52 and 78 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) eight,

 (select distinct a.tag_id as tag_id,
    (select count(b.timestamp) as leftDefMid
     from match_tromso_tottenham_2013_uefa_r13  b
     where b.position[0] between 52 and 78 AND
     '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
     b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id) nine,

  (select distinct a.tag_id as tag_id,
     (select count(b.timestamp) as rightDef
      from match_tromso_tottenham_2013_uefa_r13  b
      where b.position[0] between 78 and 105 AND
      '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
      b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) ten,
```

```
 (select distinct a.tag_id as tag_id,
     (select count(b.timestamp) as centerDef
      from match_tromso_tottenham_2013_uefa_r13  b
      where b.position[0] between 78 and 105 AND
      '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
      b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) eleven,

 (select distinct a.tag_id as tag_id,
     (select count(b.timestamp) as leftDef
      from match_tromso_tottenham_2013_uefa_r13  b
      where b.position[0] between 78 and 105 AND
      '2013-11-28 19:03:51'>(b.timestamp+interval '-55' minute) AND
      b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id) twelwe

 where one.tag_id = two.tag_id AND two.tag_id = three.tag_id AND
       three.tag_id = four.tag_id AND four.tag_id = five.tag_id AND
       five.tag_id = six.tag_id AND six.tag_id = seven.tag_id AND
       seven.tag_id =eight.tag_id AND eight.tag_id= nine.tag_id AND
       nine.tag_id = ten.tag_id AND ten.tag_id = eleven.tag_id AND
       eleven.tag_id = twelwe.tag_id
 ORDER BY one.tag_id;
```

Listing 22: Eksempel 2 - *Hyppige posisjoner*

```
explain analyze
with one AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as rightAttack
   from match_tromso_tottenham_2013_uefa_r13 b
   where a.tag_id = b.tag_id AND b.position[0] between 0 and 26 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 50 and 69)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by a.tag_id),
two AS
```

```
  (select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerAttack
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 0 and 26 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id),
three AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as leftAttack
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 0 and 26 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
 from match_tromso_tottenham_2013_uefa_r13 a
 group by tag_id),
four AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightAttackMid
   from match_tromso_tottenham_2013_uefa_r13 b
   where b.position[0] between 26 and 52 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 50 and 69 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
five AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerAttackMid
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 26 and 52 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
six AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ leftAttackMid
```

```
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 26 and 52 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
seven AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as rightDefMid
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 52 and 78 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
eight AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerDefMid
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 52 and 78 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
nine AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as leftDefMid
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 52 and 78 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
ten AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as rightDef
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 78 and 105 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
```

```
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
eleven AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as centerDef
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 78 and 105 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id),
twelwe AS
  (select a.tag_id as tag_id, (select count(b.timestamp) as leftDef
   from match_tromso_tottenham_2013_uefa_r13  b
   where b.position[0] between 78 and 105 AND
   '2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
   b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
  from match_tromso_tottenham_2013_uefa_r13 a
  group by tag_id)

SELECT distinct one.tag_id as tag_id, one.rightAttack as rightAttack,
       two.centerAttack as centerAttack, three.leftAttack as leftAttack
 ↪ , four.rightAttackMid as rightAttackMid,
       five.centerAttackMid as centerAttackMid, six.leftAttackMid as
 ↪ leftAttackMid, seven.rightDefMid as rightDefMid,
       eight.centerDefMid as centerDefMid, nine.leftDefMid as
 ↪ leftDefMid, ten.rightDef as rightDef, eleven.centerDef as centerDef
 ↪ ,
       twelwe.leftDef as leftDef, (one.rightAttack + two.centerAttack +
 ↪ three.leftAttack +four.rightAttackMid+ five.centerAttackMid +
       six.leftAttackMid + seven.rightDefMid + eight.centerDefMid +
 ↪ nine.leftDefMid + ten.rightDef + eleven.centerDef + twelwe.leftDef)
 ↪  as sum
FROM one, two, three, four, five, six, seven, eight, nine, ten, eleven,
 ↪  twelwe
where one.tag_id = two.tag_id AND two.tag_id = three.tag_id AND three.
 ↪ tag_id = four.tag_id AND
     four.tag_id = five.tag_id AND five.tag_id = six.tag_id AND six.
```

```
 ↪ tag_id = seven.tag_id AND
      seven.tag_id =eight.tag_id AND eight.tag_id= nine.tag_id AND nine
 ↪ .tag_id = ten.tag_id AND ten.tag_id = eleven.tag_id AND
      eleven.tag_id = twelwe.tag_id
ORDER BY one.tag_id;
```

Listing 23: Eksempel 3 - *Hyppige posisjoner*

```
explain analyze
SELECT distinct one.tag_id as tag_id, one.rightAttack as rightAttack,
two.centerAttack as centerAttack, three.leftAttack as leftAttack, four.
 ↪ rightAttackMid as rightAttackMid,
five.centerAttackMid as centerAttackMid, six.leftAttackMid as
 ↪ leftAttackMid, seven.rightDefMid as rightDefMid,
eight.centerDefMid as centerDefMid, nine.leftDefMid as leftDefMid, ten.
 ↪ rightDef as rightDef, eleven.centerDef as centerDef,
twelwe.leftDef as leftDef, (one.rightAttack + two.centerAttack +three.
 ↪ leftAttack +four.rightAttackMid+ five.centerAttackMid +
six.leftAttackMid + seven.rightDefMid + eight.centerDefMid + nine.
 ↪ leftDefMid + ten.rightDef + eleven.centerDef + twelwe.leftDef) as
 ↪ sum
FROM

(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightAttack
from match_tromso_tottenham_2013_uefa_r13 b
where a.tag_id = b.tag_id AND b.position[0] between 0 and 26 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 69)
from match_tromso_tottenham_2013_uefa_r13 a
group by a.tag_id) one
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerAttack
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 0 and 26 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
```

```
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) two
ON one.tag_id = two.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ leftAttack
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 0 and 26 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) three
ON two.tag_id = three.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightAttackMid
from match_tromso_tottenham_2013_uefa_r13 b
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 69 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) four
ON three.tag_id = four.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerAttackMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) five
ON four.tag_id = five.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ leftAttackMid
from match_tromso_tottenham_2013_uefa_r13  b
```

```
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) six
ON five.tag_id = six.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) seven
ON six.tag_id = seven.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) eight
ON seven.tag_id = eight.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ leftDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) nine
ON eight.tag_id = nine.tag_id
JOIN
```

```
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) ten
ON nine.tag_id = ten.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) eleven
ON ten.tag_id = eleven.tag_id
JOIN
(select distinct a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ leftDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id) twelwe
ON eleven.tag_id = twelwe.tag_id
ORDER BY one.tag_id;
```

Listing 24: Eksempel 4 - *Hyppige posisjoner*

```
explain analyze
with one AS
(select a.tag_id as tag_id, (select count(b.timestamp) as rightAttack
from match_tromso_tottenham_2013_uefa_r13 b
where a.tag_id = b.tag_id AND b.position[0] between 0 and 26 AND
```

```
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 69)
from match_tromso_tottenham_2013_uefa_r13 a
group by a.tag_id),
two AS
(select a.tag_id as tag_id, (select count(b.timestamp) as centerAttack
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 0 and 26 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
three AS
(select a.tag_id as tag_id, (select count(b.timestamp) as leftAttack
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 0 and 26 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
four AS
(select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ rightAttackMid
from match_tromso_tottenham_2013_uefa_r13 b
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 69 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
five AS
(select a.tag_id as tag_id, (select count(b.timestamp) as
 ↪ centerAttackMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
```

```
group by tag_id),
six AS
(select a.tag_id as tag_id, (select count(b.timestamp) as leftAttackMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 26 and 52 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
seven AS
(select a.tag_id as tag_id, (select count(b.timestamp) as rightDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
eight AS
(select a.tag_id as tag_id, (select count(b.timestamp) as centerDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
nine AS
(select a.tag_id as tag_id, (select count(b.timestamp) as leftDefMid
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 52 and 78 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
ten AS
(select a.tag_id as tag_id, (select count(b.timestamp) as rightDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
```

```
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 50 and 68 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
eleven AS
(select a.tag_id as tag_id, (select count(b.timestamp) as centerDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 20 and 50 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id),
twelwe AS
(select a.tag_id as tag_id, (select count(b.timestamp) as leftDef
from match_tromso_tottenham_2013_uefa_r13  b
where b.position[0] between 78 and 105 AND
'2013-11-28 19:03:51' >(b.timestamp + interval '-55' minute) AND
b.position[1] between 0 and 20 AND a.tag_id = b.tag_id)
from match_tromso_tottenham_2013_uefa_r13 a
group by tag_id)

SELECT distinct one.tag_id as tag_id, one.rightAttack as rightAttack,
two.centerAttack as centerAttack, three.leftAttack as leftAttack, four.
 ↪ rightAttackMid as rightAttackMid,
five.centerAttackMid as centerAttackMid, six.leftAttackMid as
 ↪ leftAttackMid, seven.rightDefMid as rightDefMid,
eight.centerDefMid as centerDefMid, nine.leftDefMid as leftDefMid, ten.
 ↪ rightDef as rightDef, eleven.centerDef as centerDef,
twelwe.leftDef as leftDef, (one.rightAttack + two.centerAttack +three.
 ↪ leftAttack +four.rightAttackMid+ five.centerAttackMid +
six.leftAttackMid + seven.rightDefMid + eight.centerDefMid + nine.
 ↪ leftDefMid + ten.rightDef + eleven.centerDef + twelwe.leftDef) as
 ↪ sum
FROM one
JOIN two
ON one.tag_id = two.tag_id
JOIN three
```

```
ON two.tag_id = three.tag_id
JOIN four
ON three.tag_id = four.tag_id
JOIN five
ON four.tag_id = five.tag_id
JOIN six
ON five.tag_id = six.tag_id
JOIN seven
ON six.tag_id = seven.tag_id
JOIN eight
ON seven.tag_id = eight.tag_id
JOIN nine
ON eight.tag_id = nine.tag_id
JOIN ten
ON nine.tag_id = ten.tag_id
JOIN eleven
ON ten.tag_id = eleven.tag_id
JOIN twelwe
ON eleven.tag_id = twelwe.tag_id
ORDER BY one.tag_id;
```

Vedlegg G

Kildekoden til det implementerte analyseverktøyet, kan hentes via følgene URL: *https://bitbucket.org/snir0803/bagadus*. Tilgang gis etter forespørsel.

Bibliografi

[1] AngularJS. *routeProvider*. Tilgjengelig fra https://docs.angularjs.org/api/ngRoute/provider/$routeProvider (besøkt 13.05.2015).

[2] Angularjs. *timeout*. Tilgjengelig fra https://docs.angularjs.org/api/ng/service/$timeout (besøkt 13.05.2015).

[3] AngularJS. *What are Scopes*. Tilgjengelig fra https://docs.angularjs.org/guide/scope (besøkt 13.05.2015).

[4] Angularjs. *What Is Angular*. Tilgjengelig fra https://docs.angularjs.org/guide/introduction (besøkt 13.05.2015).

[5] Apache. *Welcome to Apache Maven*. Tilgjengelig fra http://maven.apache.org/ (besøkt 13.05.2015).

[6] Dinesh Asanka. *Whether to use UNION or OR in SQL Server Queries*. Red. av SQL Server Performance. Tilgjengelig fra http://www.sql-server-performance.com/2011/union-or-sql-server-queries/ (besøkt 13.05.2015). 2011.

[7] Basler. *ace Series*. Tilgjengelig fra http://www.baslerweb.com/en/products/area-scan-cameras/ace (besøkt 13.05.2015).

[8] Youssef Bassil. «A comparative study on the performance of the Top DBMS systems». I: *arXiv preprint arXiv:1205.2889* (2012).

[9] Redback Biotek. Tilgjengelig fra http://www.redbackbiotek.com/sports-gps-tracking (besøkt 13.05.2015).

[10] Mads Bøyum. *Hanstveit mener Brann løp for lite*. Red. av folkebladet.no. Tilgjengelig fra: http://www.folkebladet.no/100Sport/fotball/eliteserien/article463235.snd. [Online; publisert 19-september-2014] (besøkt 13.05.2015). 2014.

[11] Surajit Chaudhuri. «An Overview of Query Optimization in Relational Systems». I: *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. PODS '98. Seattle, Washington, USA: ACM, 1998, s. 34–43. ISBN: 0-89791-996-3. DOI: 10.1145/275487.275492. URL: http://doi.acm.org/10.1145/275487.275492.

[12] Surajit Chaudhuri og Kyuseok Shim. «Including group-by in query optimization». I: *VLDB*. Bd. 94. 1994, s. 354–366.

[13] Clansoft. *Home*. Tilgjengelig fra https://www.tacticalpad.com/tacticalpad/index.php(besøkt 13.05.2015).

[14] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, Paul R. Young. «Computing As a Discipline». I: *Commun. ACM* 32.1 (jan. 1989). Red. av Peter J. Denning, s. 9–23. ISSN: 0001-0782. DOI: 10.1145/63238.63239. URL: http://doi.acm.org/10.1145/63238.63239.

[15] Atlassian Developers. *Bitbucket*. Tilgjengelig fra https://www.atlassian.com/software/bitbucket/overview (besøkt 13.05.2015).

[16] Business Dictionary. *heatmap*. Tilgjengelig fra http://www.businessdictionary.com/definition/heatmap.html (besøkt 13.05.2015).

[17] Derek Dieter. *Using union instead of or*. Red. av sqlserverplanet.com. Tilgjengelig fra: http://sqlserverplanet.com/optimization/using-union-instead-of-or. [Online; publisert 7-februar-2012] (besøkt 13.05.2015). 2012.

[18] Eclipse. *Jetty*. Tilgjengelig fra http://www.eclipse.org/jetty/ (besøkt 13.04.2015).

[19] FFmpeg. *About FFmpeg*. Tilgjengelig fra https://www.ffmpeg.org/about.html (besøkt 13.05.2015).

[20] Fluendo. *Longomatch*. Tilgjengelig fra http://longomatch.com/index.html (besøkt 13.05.2015).

[21] jQuery Foundation. *What is jQuery?* Tilgjengelig fra http://jquery.com/ (besøkt 13.05.2015).

[22] Vamsidhar Reddy Gaddam, Ragnar Langseth, Håkon Kvalse Stenslands, Carsten Griwodz, Pål Halvorsen, Øystein Landsverk. «Automatic Real-Time Zooming and Panning on Salient Objects From a Panoramic Video». I: *Proceedings of the ACM International Conference on Multimedia (ACM MM)*. Orlando, FL, USA, 2014, s. 725–726.

[23] Vamsidhar Reddy Gaddam, Ragnar Langseth, Håkon Kvale Stensland, Pierre Gurdjos, Vincent Charvillat, Carsten Griwodz, Dag Johansen, Pål Halvorsen. «Be Your Own Cameraman: Real-time Support for Zooming and Panning into Stored and Live Panoramic Video». I: *Proceedings of the 5th ACM Multimedia Systems Conference*. MMSys '14. Singapore, Singapore: ACM, 2014, s. 168–171. ISBN: 978-1-4503-2705-3. DOI: 10.1145/2557642.2579370. URL: http://doi.acm.org/10.1145/2557642.2579370.

[24] Vamsidhar Reddy Gaddam, Ragnar Langseth, Sigurd Ljødal, Pierre Gurdjos, Vincent Charvillat, Carsten Griwodz, Pål Halvorsen. «Interactive Zoom and Panning from Live Panoramic Video». I: *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. Singapore, 2014.

[25] GPSports. *GPSports*. Tilgjengelig fra http://gpsports.com/ (besøkt 13.05.2015).

[26] The PostgreSQL Global Development Group. *About Postgresql*. Tilgjengelig fra http://www.postgresql.org/about/ (besøkt 13.05.2015).

[27] Pål Halvorsen, Simen Sægrov, Asgeir Mortensen, David K. C. Kristensen, Alexander Eichhorn, Magnus Stenhaug, Stian Dahl, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Dag Griwodz Carsten og Johansen. «Bagadus: An Integrated System for Arena Sports Analytics - A Soccer Case Study». I: *Proceedings of the 4th ACM Multimedia Systems Conference*. Oslo, Norway, 2013, s. 48–59.

[28] Match Analysis Inc. Tilgjengelig fra http://matchanalysis.com/ (besøkt 13.05.2015).

[29] Stack Exchange Inc. *SQL JOIN: is there a difference between USING, ON or WHERE?* Tilgjengelig fra http://stackoverflow.com/questions/5654278/sql-join-is-there-a-difference-between-using-on-or-where/5654338 (besøkt 13.05.2015).

[30] Stack Exchange Inc. *What is more efficient, a where clause or a join with million plus row tables?* Tilgjengelig fra http://dba.stackexchange.com/questions/3480/what-is-more-efficient-a-where-clause-or-a-join-with-million-plus-row-tables (besøkt 13.05.2015).

[31] Sports Interactive. *Game Library*. Tilgjengelig fra http://www.sigames.com/game-library (besøkt 13.05.2015).

[32] Jetbrains. *Intellij IDEA*. Tilgjengelig fra https://www.jetbrains.com/idea/ (besøkt 13.05.2015).

[33] Dag Johansen, Magnus Stenhaug, Roger B. Hansen, Agnar Christensen, Per Mathias Høgmo. «Muithu: Smaller Footprint, Potentially Larger Imprint». I: *Digital Information Management (ICDIM), 2012 Seventh International Conference on. IEEE.* 2012, s. 205–214.

[34] JSON. *Introducing JSON*. Tilgjengelig fra `http://www.json.org/` (besøkt 13.05.2015).

[35] Kaazing. *What is WebSocket?* Tilgjengelig fra `http://kaazing.com/websocket/` (besøkt 13.05.2015).

[36] Archana P Kumar, Abhishekh Kumar og Vipin N Kumar. «A Comprehensive Comparative study of SPARQL and SQL». I: *International Journal of Computer Science and Information Technologies* 2.4 (2011), s. 1706–1710.

[37] Gregory Larsen. *Using a Subquery in a T-SQL Statement*. Red. av Database Journal. Tilgjengelig fra `http://www.databasejournal.com/features/mssql/article.php/3464481/Using-a-Subquery-in-a-T-SQL-Statement.htm` (besøkt 13.05.2015). 2005.

[38] Deep Liquid. *Jcrop - the jQuery Image Cropping Plugin*. Tilgjengelig fra `http://deepliquid.com/content/Jcrop.html` (besøkt 13.05.2015).

[39] Stats LLC. *Coaching*. Tilgjengelig fra `http://www.sportvu.com/football_coaching.asp` (besøkt 13.05.2015).

[40] Microsoft. *SQL Server*. Tilgjengelig fra `http://www.microsoft.com/en-us/server-cloud/products/sql-server/` (besøkt 13.05.2015). 2015.

[41] Asgeir Mortensen, Vamsidhar Reddy Gaddam, Håkon Kvale Stensland, Carsten Griwodz, Dag Johansen, Pål Halvorsen. «Automatic Event Extraction and Video Summaries from Soccer Games». I: *Proceedings of the 5th ACM Multimedia Systems Conference*. MMSys '14. Singapore, Singapore: ACM, 2014, s. 176–179. ISBN: 978-1-4503-2705-3. DOI: `10.1145/2557642.2579374`. URL: `http://doi.acm.org/10.1145/2557642.2579374`.

[42] Ellen Munthe-Kaas. *Effektiv eksekvering av spørsmål*. Forelesning. Tilgjengelig fra `http://www.uio.no/studier/emner/matnat/ifi/INF3100/v15/undervisningsmateriale/forelesningsmateriale/kap15.pdf` (besøkt 13.05.2015). 2015.

[43] OpenGL. *Compute Shader*. Tilgjengelig fra `https://www.opengl.org/wiki/Compute_Shader` (besøkt 13.05.2015).

[44] Opta. *about opta*. Tilgjengelig fra `http://www.optasports.com/about/who-we-are/about-opta.aspx` (besøkt 13.05.2015).

[45] Opta. *Football Matchday Widget Showcase*. Tilgjengelig fra http://www.optasports.com/football-matchday-widget-showcase.aspx (besøkt 13.05.2015).

[46] Oracle. *Class DriverManager*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/sql/DriverManager.html (besøkt 13.05.2015).

[47] Oracle. *Class String*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/lang/String.html (besøkt 13.05.2015).

[48] Oracle. *Class TreeMap<K,V>*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html (besøkt 13.05.2015).

[49] Oracle. *Interface Connection*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html (besøkt 13.05.2015).

[50] Oracle. *Interface ResultSet*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html (besøkt 29.11.2014).

[51] Oracle. *Interface Statement*. Tilgjengelig fra http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html (besøkt 13.05.2015).

[52] Oracle. *Java SE Technologies - Database*. Tilgjengelig fra http://www.oracle.com/technetwork/java/javase/jdbc/index.html (besøkt 13.05.2015).

[53] Mark Otto og Jacob Thornton. *About Bootstrap*. Tilgjengelig fra http://getbootstrap.com/about/ (besøkt 13.05.2015).

[54] Svein Arne Pettersen, Dag Johansen, Håvard Johansen, Vegard Berg-Johansen, Vamsidhar Reddy Gaddam, Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, Håkon Kvale Stensland, Pål Halvorsen. «Soccer Video and Player Position Dataset». I: *Proceedings of the 5th ACM Multimedia Systems Conference*. MMSys '14. Singapore, Singapore: ACM, 2014, s. 18–23. ISBN: 978-1-4503-2705-3. DOI: 10.1145/2557642.2563677. URL: http://doi.acm.org/10.1145/2557642.2563677.

[55] Planetmath. *Cartesian coordinates*. Tilgjengelig fra http://planetmath.org/cartesiancoordinates (besøkt 13.05.2015).

[56] PostgeSQL. *Aggregate Functions*. Tilgjengelig fra http://www.postgresql.org/docs/9.1/static/functions-aggregate.html (besøkt 13.05.2015).

[57] PostgeSQL. *Date/Time Functions and Operators*. Tilgjengelig fra http://www.postgresql.org/docs/9.1/static/functions-datetime.html (besøkt 13.05.2015).

[58] PostgreSQL. *Combining queries*. Tilgjengelig fra http://www.postgresql.org/docs/8.3/static/queries-union.html (besøkt 13.15.2015).

[59] PostgreSQL. *Create function*. Tilgjengelig fra http://www.postgresql.org/docs/9.1/static/sql-createfunction.html (besøkt 13.05.2015).

[60] PostgreSQL. *Explain*. Tilgjengelig fra http://www.postgresql.org/docs/9.1/static/sql-explain.html (besøkt 13.05.2015).

[61] Postgresql. *Geometric Functions and Operators*. Tilgjengelig fra http://www.postgresql.org/docs/9.1/static/functions-geometry.html (besøkt 13.05.2015).

[62] PostgreSQL. *Joins between tables*. Tilgjengelig fra http://www.postgresql.org/docs/8.1/static/tutorial-join.html (besøkt 13.05.2015).

[63] PostgreSQL. *Select*. Tilgjengelig fra http://www.postgresql.org/docs/9.0/static/sql-select.html (besøkt 13.05.2015).

[64] PostgreSQL. *WITH Queries*. Tilgjengelig fra http://www.postgresql.org/docs/8.4/static/queries-with.html (besøkt 13.05.2015).

[65] Prozone. *Prozone Sports Performance Analysis*. Tilgjengelig fra http://www.prozonesports.com/ (besøkt 13.05.2015).

[66] Lars Kristian Roland. *Hibernate*. Tilgjengelig fra www.uio.no/studier/emner/matnat/ifi/INF5750/h13/lecture-presentations/inf5750---lecture-2.-c---hibernate-intro.pdf (besøkt 13.04.2015). 2013.

[67] Lars Kristian Roland. *Maven*. Forelesning. Tilgjengelig fra http://www.uio.no/studier/emner/matnat/ifi/INF5750/h13/lecture-presentations/inf5750---lecture-1---b-maven.pdf (besøkt 15.05.2015). 2013.

[68] Lars Kristian Roland. *Spring Framework Basics*. Forelesning. Tilgjengelig fra http://www.uio.no/studier/emner/matnat/ifi/INF5750/h13/lecture-presentations/inf5750---lecture-2.-a---spring-framework-basics.pdf (besøkt 13.05.2015). 2013.

[69] Interplay Sports. *Interplay Sports*. Tilgjengelig fra http://www.interplay-sports.com/ (besøkt 13.05.2015).

[70] Spring. *Annotation Type PathVariable*. Tilgjengelig fra http://docs.spring.io/spring-framework/docs/4.1.x/javadoc-api/org/springframework/web/bind/annotation/PathVariable.html (besøkt 13.05.2015).

[71] Spring. *Annotation Type RequestMapping*. Tilgjengelig fra http://docs.spring.io/spring/docs/2.5.x/api/org/springframework/web/bind/annotation/RequestMapping.html (besøkt 13.05.2015).

[72] Spring. *Annotation Type Service*. Tilgjengelig fra http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Service.html (besøkt 13.05.2015).

[73] Spring. *Spring Framework*. Tilgjengelig fra https://spring.io/ (besøkt 13.05.2015).

[74] Spring. *Understanding REST*. Tilgjengelig fra http://spring.io/understanding/REST (besøkt 13.04.2015).

[75] StatSports. *StatSports*. Tilgjengelig fra http://statsports.ie/ (besøkt 13.05.2015).

[76] Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Marius Tennøe, Espen Helgedagsrud, Mikkel Næss, Henrik Kjus Alstad, Asgeir Mortensen, Ragnar Langseth, Sigurd Ljødal, Østein Landsverk, Carsten Griwodz, Pål Halvorsen, Magnus Stenhaug, Dag Johansen. «Bagadus: An Integrated Real-time System for Soccer Analytics». I: *ACM Trans. Multimedia Comput. Commun. Appl.* 10.1s (jan. 2014), 14:1–14:21. ISSN: 1551-6857. DOI: 10.1145/2541011. URL: http://doi.acm.org/10.1145/2541011.

[77] Daniel Stuparu og Manuela Petrescu. «Common Table Expression: Different Database Systems Approach». I: *Journal of Communication and Computer* 6.3 (2009), s. 9–15.

[78] Truls Grane Sylvarnes. *-Van Persie vurderer United-fremtiden*. Red. av TV2.no. Tilgjengelig fra: http://www.tv2.no/a/5291138. [Online; publisert 27-februar-2014] (besøkt 03.05.2015). 2014.

[79] Simen Sægrov, Eichhorn Alexander, Jørgen Emerslund, Håkon Kvale Stensland, Carsten Griwodz, Dag Johansen, Halvorsen Pål. «Bagadus: An integrated system for soccer analysis (demo)». I: *Proceedings of the International Conference on Distributed Smart Cameras (ICDSC)*. 2012.

[80] Christer Sævig. *Høgmo skal overvåke spillerne på trening*. Red. av TV2.no. Tilgjengelig fra: http://www.tv2.no/a/6182222. [Online; publisert 31-oktober-2014] (besøkt 13.05.2015). 2014.

[81] TechTerms. *Tooltip*. Tilgjengelig fra http://www.techterms.com/definition/tooltip (besøkt 13.05.2015).

[82] ZXY Sport Tracking. *ZXY Sport Tracking*. Tilgjengelig fra http://www.zxy.no/ (besøkt 13.05.2015).

[83] Bjarte Valen. *Genistreken som fikk fart på United*. Red. av united.no. Tilgjengelig fra: http://www.united.no/nyhetsarkiv/genistreken-som-fikk-fart-pa-united/. [Online; publisert 26-mars-2015]. 2015.

[84] VideoLan. *x264*. Tilgjengelig fra http://www.videolan.org/developers/x264.html (besøkt 13.05.2015).

[85] w3schools. *HTML <select> tag*. Tilgjengelig fra http://www.w3schools.com/tags/tag_select.asp (besøkt 13.05.2015).

[86] w3schools. *JavaScript Events*. Tilgjengelig fra `http://www.w3schools.com/html/html5_canvas.asp` (besøkt 13.05.2015).

[87] w3schools. *SQL count() function*. Tilgjengelig fra `http://www.w3schools.com/sql/sql_func_count.asp` (besøkt 13.05.2015).

[88] w3schools. *SQL having clause*. Tilgjengelig fra `http://www.w3schools.com/sql/sql_having.asp` (besøkt 13.05.2015).

[89] w3schools. *SQL max() function*. Tilgjengelig fra `http://www.w3schools.com/sql/sql_func_max.asp` (besøkt 13.05.2015).

[90] w3schools. *SQL WHERE Clause*. Tilgjengelig fra `http://www.w3schools.com/sql/sql_where.asp` (besøkt 13.05.2015).

[91] w3schools. *The HTML <canvas> Element*. Tilgjengelig fra `http://www.w3schools.com/html/html5_canvas.asp` (besøkt 13.05.2015).

[92] Matt Walsh. *Is It Time For Spurs To Start Their £30m Man? | Erik Lamela Analysis*. Red. av EPL index. Tilgjengelig fra: `http://eplindex.com/44624/time-spurs-start-30m-man-erik-lamela-analysis.html`. [Online; publisert 19-november-2013] (besøkt 13.05.2015). 2013.

[93] Martin Aleksander Wilhelmsen, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam, Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, Pål Halvorsen. «Using a Commodity Hardware Video Encoder for Interactive Video Streaming». I: *Proceedings of the IEEE International Symposium on Multimedia*. Taichung, Taiwan: IEEE, 2014.

[94] Youtube. *Barcelona 1 - 2 Real Madrid, Cristiano ronaldo Goal ! HD*. Tilgjengelig fra `https://www.youtube.com/watch?v=7LUFC0Skq3U` (besøkt 13.05.2015).

[95] Nvidia cuda zone. *NVIDIA VIDEO CODEC SDK*. Tilgjengelig fra `https://developer.nvidia.com/nvidia-video-codec-sdk` (besøkt 13.05.2015).